A Dynamic Stroke Segmentation Technique for Sketched Symbol Recognition

Vincenzo Deufemia and Michele Risi

Dipartimento di Matematica e Informatica
Università di Salerno,
84084 Fisciano (SA), Italy
{deufemia,mrisi}@unisa.it

Abstract. In this paper, we address the problem of ink parsing, which tries to identify distinct symbols from a stream of pen strokes. An important task of this process is the segmentation of the users' pen strokes into salient fragments based on geometric features. This process allows users to create a sketch symbol varying the number of pen strokes, obtaining a more natural drawing environment. The proposed sketch recognition technique is an extension of LR parsing techniques, and includes ink segmentation and context disambiguation. During the parsing process, the strokes are incrementally segmented by using a dynamic programming algorithm. The segmentation process is based on templates specified in the productions of the grammar specification from which the parser is automatically constructed.

1 Introduction

Sketches greatly simplify conceptual design activities through abstract models that let designers express their creativeness, and focus on critical issues rather than on intricate details [9]. Due to their minimalist nature, i.e., representing only what is necessary, they enhance collaboration and communication efficiency.

Underlying a sketch-based user interface several processes can be activated. These include the processing of pen strokes, recognition of symbols, stroke beautification, reasoning about shapes, and high-level interpretation. The sketch understanding tasks are not trivial because recognizing the meaningful patterns implied by a user's pen stroke must be flexible enough to allow some tolerance in sketch recognition, but sufficiently constrained not to accept incorrect patterns. Furthermore, the context in which a particular stroke or group of strokes appears considerably influences the interpretation of that stroke. From a visual language point of view, this means that the interpretation of a graphical object is strongly influenced by the objects surrounding it. Moreover, semantically different objects might be graphically represented by identical or apparently similar symbols.

Another important issue in sketch understanding concerns ink parsing, which refers to the task of grouping and segmenting the user's strokes into clusters of intended symbols. This allows users to create a sketch symbol varying the number of pen strokes, obtaining a more natural drawing environment.

J.S. Marques et al. (Eds.): IbPRIA 2005, LNCS 3523, pp. 328–335, 2005.

Several systems for sketch recognition constrain users to draw an entire symbol as single stroke [8,9], or to draw only strokes representing single primitive shapes such as lines, arcs, or curves [7,14]. In other systems prior to parsing, the input sketch is segmented into line and arc segments allowing symbols to be drawn with multiple pen strokes, and a single pen stroke to contain multiple symbols [2,11]. Different approaches to segmentation have been proposed, some systems segment the strokes by using their curvature and speed information [2,11], Saund uses both local features (such as intersections and curvatures) and global features (such as closed paths) to locate breakpoints of a stroke [12], whereas Yu applied the mean shift procedure to approximate strokes [13]. Hse *et al.* [6] have presented an optimal segmentation approach based on template that does not suffer of over- and under-segmentation of strokes. In particular, given a sketched symbol S and a template T, the algorithm finds a set of breakpoints in S such that the fitting performed according to T yields the minimum fit error. The templates T can be of two types, one specifies a sequence of lines and ellipses and the other specifies the number of lines and ellipses.

Segmentation is a basic problem that has many applications for digital ink capture and manipulation, as well as higher-level symbolic and structural analyses. As an example, the structural information generated by the segmentation process can be useful for the beautification of the symbols [7], for developing a user interface with which to interact with sketched ink.

In this paper, we present a sketch recognition technique that takes into account both the problem of context based ambiguity resolution and ink segmentation. The sketch parser relies on an extension of LR parsing techniques and is automatically generated from a grammar specification. The proposed segmentation technique dynamically segments the strokes during the parsing process by using an extended version of the optimal pen strokes segmentation technique proposed by Hse *et al.* [6]. The template given in input to the algorithm is a sequence of primitive shapes iteratively extracted from grammar productions. Thus, the parser's context drives the segmentation process of the strokes.

The paper is organized as follows. We first discuss the formalism for describing sketch languages, and the parsing approach underlying the proposed framework. Successively, we present a dynamic segmentation technique that integrated into the parsing algorithm solves the problem of multi-stroke recognition. In section 4 we describe an example of application of the parsing algorithm on hand-drawn circuit diagrams. Finally, the conclusion and further research are discussed in Section 5.

2 A Grammar-Based Sketch Parsing Approach

Because we use an extension of LR parsing technique [1], we describe sketch languages using the formalism of *eXtended Positional Grammars* (XPG, for short) [4]. XPGs represent a direct extension of context-free string grammars, where more general relations other than concatenation are allowed. A sentence is conceived as a set of symbols with attributes. Such attributes are also determined by the relationships holding among the symbols. Thus, a sentence is specified by combining symbols with relations. In particular, the productions have the following format:

$$A \rightarrow x_1 \boldsymbol{R_1} x_2 \boldsymbol{R_2} \ldots x_{m-1} \boldsymbol{R_{m-1}} x_m$$

where each $\boldsymbol{R_j}$ define a sequence of relation between x_{j+1} and x_{j-i}, with $1 \leq i < j$, by means of a threshold t_j.

An XPG for modeling a sketch language L can be logically partitioned into two XPG grammars. The first, named *ink grammar*, defines the symbols of the language L as geometric compositions of primitive objects, i.e., patterns that cannot be recognized as a combination of other objects and must be recognized directly, such as line segments and elliptical arcs. For example, a production specifying an *Arrow* symbol is shown in the bottom of Figure 1 together with a sketch matching such production. The second grammar, named *language grammar*, specifies the sentences of the language as compositions of shapes defined in the ink grammar through spatial relations. The production at the top of Figure 1 defines a *SubCircuit* object as the composition of three language symbols: *Gain*, *Arrow* and *Sum*, and shows a sketch matching the production.

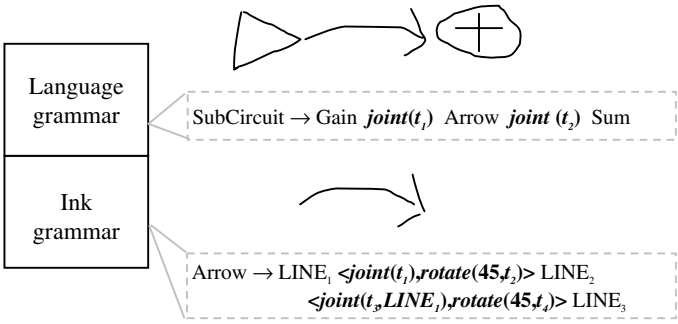

Fig. 1. The two levels of XPG grammar specification.

The recognition of line segments is performed with the least square fitting [5], whereas the elliptical arc fitting is performed with the technique proposed in [10].

The definition of XPGs has been strongly influenced by the need of having an efficient parser able to process the generated languages. Due to their analogy with string grammars, it has been natural the use of LR parsing techniques [4]. The result was a parser that scans the input in a non-sequential way, driven by the relations used in the grammar. In order to guide the scanning of the input symbols, a new column next is added to the usual action and goto parts of an LR parsing table. For each state of the parser, this column contains an entry with information to access the next symbol to be parsed. This information is derived by the relations of the grammar productions, during the parser generation. Thus, during the parsing process, a sketch parser generates a sequence of calls to a function *Fetch_Stroke* that linearizes the input at run-time [3].

Figure 2 shows the structure of the recognition process. During the parsing of a sketch diagram, a rank value is computed by combining the accuracy of the strokes forming the sketch and of their spatial relations. Thus, the output of the parser is a *probabilistic parse forest* where each tree in corresponds to an interpretation of the

sketch sentence. Each node of a tree has associated a probability representing the rank of the stroke interpretation associated to the leaves of its subtree. Such trees can be analyzed to obtain a rank of the interpretations by considering the probability associated to the roots of the trees and the number of language symbols recognized, similarly to what done for natural languages.

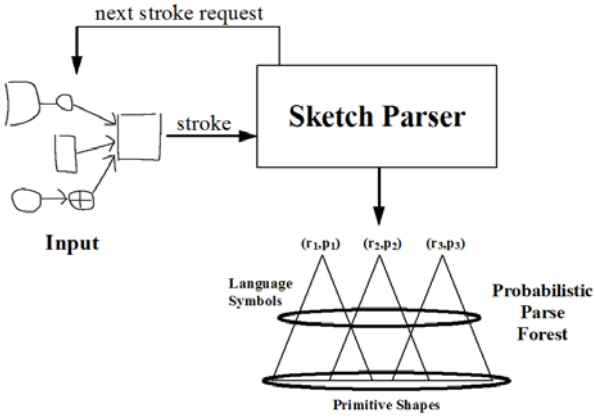

Fig. 2. The sketch parsing approach.

3 A Dynamic Ink Segmentation Technique

The previous sketch parsing approach does not allow pen strokes to represent any number of shape primitives connected together, since the function *Fetch_Stroke* finds in the input sketch a single stroke that matches the primitive shape specified in a grammar production.

This problem can be overcome by segmenting the strokes when the parser cannot proceed in the recognition of the sketch. Thus, the sequences of primitive shapes specified in the grammar productions, together with their geometric relationships, guides the segmentation algorithm to identify the points for dividing the strokes into different primitives.

More formally, given a stroke S formed by a set of m data points, a template p formed by a sequence of primitive shapes, and an array r of relations between such shapes, the segmentation algorithm finds a subset of the m points that matches with the pattern p and the array r yielding the minimum fit error.

This "fitting to a template" problem can be optimally solved by using the dynamic programming approach proposed by Hse *et al.* [6], and considering both the similarity between segments and patterns, and the quality of the relations between the segments. Thus, the output of the algorithm is the best segmentation according to the best fit shape error and the best accuracy of the shape relations.

Let $d(m,k,p,r)$ be the minimum error segmentation, where m is the number of data points describing the stroke S, k is the number of breakpoints to be determined by the segmentation process (the start value is $k = p.len-1$), p is the template of primitive

shapes, and *r* is the array of relations specified in the XPG productions between the shapes in *p*.

The recursive definition for segmentation of *S* is given in the following.

$$d(m,k,p,r)=\begin{cases} f(S,0,m,p[1],r[1]) & \text{if } k=0 \\ \min\limits_{k<i<m}\{d(i,k-1,p[1..p.len],r[1..r.len])+f(S,i,m,p[p.len],r[r.len])\} & \text{if } k>0 \end{cases}$$

f(S,i,m,p[j],r[n]) is the *segmentation error function* defined as:

$$f(S,i,m,p,rel)=\frac{fitting(S,i,m,p)}{relation(rel,p,rel.th)}$$

This function is calculated considering the fitting of a segment from *i-th* point to *m-th* point in S using *p*, and considering the accuracy of the relation *rel* between the current matched symbol *p* and the previous one by means of the threshold *rel.th*.

When the *Fetch_Stroke* function fails to find a stroke accurately related with a previously parsed stoke, the last visited stroke *s* could contain multiple symbols. Thus, the segmentation process is activated on *s* in order to calculate the breakpoint that divide *s* into two strokes s_1 and s_2, and such that they have a good compromise between their fitting shape error and the accuracy in the relations involving them. Successively, the parser considers s_1 as the last visited stroke, and the *Fetch_Stroke* function returns s_2 as the next stroke to be parsed. The segmentation process on the stroke *s* is iterated until both *m>p.len* and *Fetch_Stroke* fails to find a stroke not in *s* and accurately related with a previously parsed stoke.

For example, a square can be drawn as a single pen stroke, or as two separate strokes, or even as three or four strokes (Fig. 3(a)-(d), respectively).

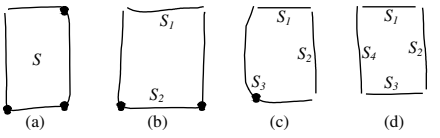

Fig. 3. Segmentation of a square.

The production used to recognize the square is:

Square → LINE_1 ***<joint(t_1), rotate(90, t_2)>*** LINE_2 ***<joint(t_1), rotate(90, t_2)>***
LINE_3 ***<joint(t_1), rotate(90, t_2), joint(t_1, LINE$_1$)>*** LINE_4

If the square is drawn using four strokes then the production is correctly reduced without segmentation since the strokes returned by *Fetch_Stroke* match the relations in the production.

During the parsing of the square in Figure 3(c), the *Fetch_Stroke* matches the first three lines of the production with the strokes S_1, S_2 and S_3, but it fails to find the symbol LINE_4. Thus, a segmentation process is activated on the last visited stroke S_3. In particular, the algorithm calculates the value *d*(*m*,1,[L,L],[{*joint*(t_1),*rotate*(90,t_2)}, {*joint*(t_1), *rotate* (90,t_2), *joint*(t_1)}]) determining the breakpoint shown as a filled circle

in Figure 3(c). The parser continues the recognition of the square by matching the segmented stroke S_3 with LINE_3 and LINE_4. Similarly, the parser recognizes the squares in Figures 3(a) and 3(b) by segmenting the strokes S and S_2 with $d(m,3,[\text{L,L,L,L}],[\varnothing,\{joint(t_1),\ rotate(90,t_2)\},\ \{joint(t_1),\ rotate(90,t_2)\},\{joint(t_1),\ rotate(90,t_2),\ joint(t_1)\}])$ and $d(m,2,[\text{L,L,L}],[\{joint(t_1),\ rotate(90,t_2)\},\{joint(t_1),rotate(90,t_2)\},\{joint(t_1),\ rotate(90,t_2),\ joint(t_1)\}])$, respectively.

It worth noting that the time and space complexity of the segmentation process is the same of the Hse's algorithm [6] since when the parser segments a stroke with the pattern $p=[p_1,\ldots,p_{n-1},p_n]$, the segmentation value of the pattern $p_r=[p_1,\ldots,p_{n-1}]$ has already been calculated previously.

4 Segmenting Hand-Drawn Electrical Circuits

In this section we show how the proposed dynamic stroke segmentation technique works during the recognition of hand-drawn circuit diagrams.

The symbols in the circuit domain are given in Figure 4.

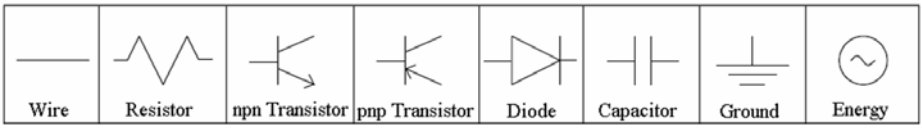

Fig. 4. The visual symbols in the circuit language.

In the following we describe some productions of the ink grammar for modeling the resistor, the wire, the capacitor and the ground symbols. In particular, the resistor symbol starts and ends with a line, and in the middle it is composed of a sequence of at least four oblique lines forming a wave.

Resistor → LINE ***<joint(t_1), rotate(225, t_2)>*** Wave ***<joint(t_1), rotate(135, t_2)>*** LINE
Wave → LINE_1 ***<joint(t_1), rotate(45, t_2)>***
LINE_2 ***<joint(t_1), rotate(135, t_2), >***
LINE_3 ***<joint(t_1), rotate(45, t_2), parallel(LINE_1, t_3)>***
LINE_4 ***<joint(t_1), rotate(135, t_2), parallel(LINE_2, t_3)>*** Multiwave
Multiwave → LINE ***<joint(t_1), rotate(45, t_2), parallel(LINE(-1), t_3)>*** Multiwave
Multiwave → LINE

Wire → Wire ***<intersect(t_1)>*** LINE
Wire → LINE

Capacitor → LINE_1 ***<perpendicular(t_1), intersection(t_2)>*** LINE_2 ***<parallel(t_3), length(1.0, t_4)>***
LINE_3 ***<perpendicular(t_1), intersection(t_2)>*** LINE_2

Ground → LINE ***<perpendicular(t_1), intersection(t_2)>*** Multiline
MultiLine → LINE ***<parallel(t_3),length(0.7, t_4), centered(t_5)>***
LINE ***<parallel(t_3),length(0.7, t_4), centered(t_5)>*** Subline
Subline → LINE ***<parallel(t_3),length(0.7, t_4), centered(t_5)>*** Subline
Subline → LINE

Notice that the LINE symbol may further be refined in order to also consider multi-stroke segments.

The following productions represent some of the language grammar productions for the circuit language.

Circuit → SubBlock
SubBlock → SubBlock ***<joint(t_i)>*** Wire ***<joint(t_i)>*** Component
SubBlock → SubBlock ***<joint(t_i)>*** Wire
SubBlock → SubBlock ***<any>*** Component
SubBlock → Component
Component → Resistor
→ Capacitor
→ Ground
→ Diode
→ pnpTransistor
→ npnTransistor
→ Energy

Fig. 5 shows an electric circuit and the segmentation of a resistor symbol. In particular, when the parser starts the recognition of the resistor it looks for a line segment. Since the resistor symbol has been drawn with two complex strokes, the line fitting algorithm fails on matching the first stroke with a line. Thus, the parser segments the stroke identifying a breakpoint that divides the stroke into two lines according to the grammar productions. The optimal breakpoint is point 1 and the parser proceeds the recognition on the remaining part of the segmented stroke. In particular, driven by the production defining the symbol *Wave* the parser segments the stroke into four line segments identifying the breakpoints 2, 3 and 4. Successively, the parser segments the second stroke forming the resistor by considering the production defining the symbol *Multiwave*.

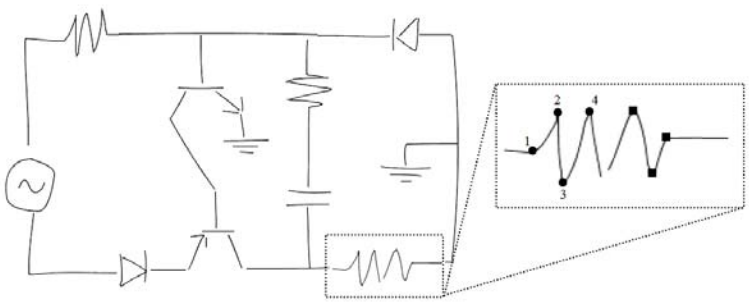

Fig. 5. An electric circuit with a resistor drawn with two strokes and segmented by the proposed algorithm during its parsing.

5 Conclusion

We have presented a sketch parsing approach that includes a context based ink segmentation technique. Indeed, template derived from grammar productions drives the

segmentation process. The approach is designed to enable natural sketch-based computer interaction since it allows for multiple symbols to be drawn in the same stroke, and allows individual symbols to be drawn in multiple strokes.

We have integrated the proposed approach in the SketchBench system [3], a tool supporting the early phases of sketch language modeling, such as shape modeling and grammar specification, and the generation of the final parser.

We are currently conducting an evaluation test of the segmentation technique.

References

1. Aho, A., Sethi, R., and Ullman, J.: Compilers Principles ,Techniques, and Tools. Addison-Wesley Series in Computer Science. 1987.
2. Calhoun, C., Stahovich, T.F., Kurtoglu, T. and Kara, L.B.: Recognizing Multi-Stroke Symbols, in AAAI Symposium - Sketch Understanding, (2002), 15–23.
3. Costagliola, G., Deufemia, V., Polese, G., and Risi, M.: A Parsing Technique for Sketch Recognition Systems, in Proceedings of IEEE Symposium VL/HCC'04, (Rome, September 2004), IEEE Press, 19–26.
4. Costagliola, G. and Polese, G.: Extended Positional Grammars, in Proceedings of IEEE Symposium VL'00, (Seattle, WA, September 2000), IEEE Press, 103–110.
5. Duda, R.O. and Hart, P.E.: Pattern Classification and Scene Analysis. Wiley Press, New York, 1973.
6. Hse, H., Shilman, M., and Newton, A.R.: Robust Sketched Symbol Fragmentation using Templates, in Proceedings of IUI'04 (Madeira, Portugal, January 2004), ACM Press, 156–160.
7. Igarashi, T., Matsuoka, S., Kawachiya, S. and Tanaka, H.: Interactive beautification: A technique for rapid geometric design, in Proceedings of UIST'97, 105–114.
8. Kimura, T.D., Apte, A., and Sengupta, S.: A graphic diagram editor for pen computers. Software Concepts and Tools, 1994, 82–95.
9. Landay, J. and Myers, B.: Sketching interfaces: Toward more human interface design. IEEE Computer, 34(3), 2001, 56–64.
10. Pilu, M., Fitzgibbon, A. and Fisher, R.: Direct Least-Square Fitting of Ellipses. IEEE Transactions on Pattern Analysis and Machine Intelligence, 21 (5), 1999, 476–480.
11. Sezgin, T.M., Stahovich, T. and Davis, R.: Sketch Based Interfaces: Early Processing for Sketch Understanding, in Procs of PUI'01, (Orlando, 2001).
12. Saund, E.: Finding Perceptually Closed Paths in Sketches and Drawings. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25 (4), 2003, 475–491.
13. Yu, B.: Recognition of Freehand Sketches using Mean Shift, in Proceedings of IUI'03, (Miami FL, 2003), 204–210.
14. Weisman, L.: A foundation for intelligent multimodal drawing and sketching programs, Master's thesis, MIT, 1999.