EDIGER - AN EDITOR FOR DIGITIZERS

J.P. Curley

Computation Centre
National Research Council of Canada

ABSTRACT

An interactive editor which records both graphical data and text is described.

From a keyboard and tablet, the user enters points, lines, and text: to any of these he may attach text labels. In subsequent editing, he forms subsets of these components based on spatial (nearness, containment), temporal (sequence) and textual criteria, and uses set operations (union, etc.) to create new sets from old. Finally, he can apply basic editing operations such as delete, move, and splice to sets as easily as to individual components.

The editor is independent of application: the file it creates has a simple format, suitable for subsequent application-specific processing.

Two quite different applications are discussed: in the first, the accurate recording of spatial data is important in itself; in the second, the spatial is more a convenience for expressing topological relationships among components in a modeling study.

RÉSUMÉ

Un appareil de mise en forme interactif qui enregistre des données graphiques et des textes est décrit.

Au moyen d'un clavier et d'une tablette, l'usager entre des points, des lignes et du texte; à chacun, il peut assigner des labels de texte. Durant une mise en forme subséquente, il forme des sous-ensembles avec ces éléments, selon des critères spatiaux (proximité, frontières) des critères temporels (séquence) et des critères se rapportant au texte, et utilise des opérations sur les ensembles (union, etc.) pour former de nouveaux ensembles. Finalement, il peut appliquer les opérations fondamentales de mise en forme comme l'effacement, le déplacement et le raccordement aussi bien aux ensembles qu'aux éléments individuels.

L'appareil de mise en forme est indépendant de l'application: le fichier créé est de format simple et convient à un traitement subséquent spécifique à l'application.

Deux applications très différentes sont décrites: dans le premier cas, l'enregistrement exact des données spatiales est important en soi, tandis que dans le second, les données spatiales sont plutôt une façon pratique d'exprimer des relations topologiques entre les éléments dans une étude de modélisation.

Introduction

A user of our central computer facility drops in and spreads a blueprint of a printed circuit board(PCB) on my desk. He needs to digitize the locations of hundreds of points in drawings such as this for subsequent drilling using a numerically controlled (NC) machine.

A second user, a researcher in the field of solar energy, asks for details on a simulation package. He describes what he wants to simulate by drawing a component diagram on a scrap of paper while he talks about parameter values for the components.

A third user, an APT programmer, arrives with a computer printout containing several pages of APT statements for a part he wishes to have machined. In a matter of minutes, he explains with a sketch what the statements "say".

This paper describes an interactive editor, "EDIGER", with potential application to areas like these, areas in which it seems natural to supplement the verbal description of an object or process with a non-verbal counterpart - a drawing. This editor records both textual and graphic data entered by the user from keyboard and digitizing tablet, and provides him with convenient facilities for modifying this information based on its spatial (proximity, containment), temporal (sequence), and textual content.

Using a computer file as the medium, EDIGER forms a machine-readable rendering of user words and actions, in the same way that a text editor renders his words alone. Like a text editor, EDIGER is independant of any specific application, but its output file format is simple enough for convenient application-specific processing.

Two applications are discussed, and related work compared.

EDIGER

EDIGER is implemented on an IBM 370/3032 under the TSS time sharing system. From a keyboard and tablet connected on a serial line to this machine, the user enters setup information - what devices he is using, menu positions, reference coordinates. He then issues the insert command to initiate recording of his subsequent actions. Five input types are recognized:

```
points
  - entered from the tablet.
text strings
  - entered from the keyboard.
LS       - line start symbol.
LE       - line end symbol.
LB <text>
 - text label to be attached to
   the previous point, text string,
   LS or LE.
```

The line-start, line-end, and label indicators are entered through menu or the keyboard as text strings commencing with a period; e.g., ".ls" indicates line start, ".lb pin22" attaches the text label "pin22" to the current record. More than one label may be attached.

Here is a short annotated input sample:

```
insert          ..user issues insert command

  component "a"  ..he keys in some text
  .ls            ..a line start symbol,
  .lb "A"        ..and a label for it
  <pt>           ..a point from tablet
  <pt>           ..second point of line
  <pt>           ..and third
  <pt>           ..and fourth
  .le            ..line-end symbol
  .lb capacity=1.2, spheat=.5
          ..and specify some parameters
```

This could be some input data for a simulation; a component is drawn as a polygon with an identifier attached to its LS symbol, and parameters attached to LE. (These conventions are samples only.)

Editing operations

Except for the coordinate transformations and menu assists described later, this recording process and its supporting program structures are similar to those of a text editor, and this similarity extends to the editing operations as well. The main differences concern the addition of spatial positioning and grouping criteria, and spatial operations for the file elements.

We will use the term record to mean a point, text string, line-start symbol, or

line-end symbol, together with its labels, that the user has entered into his file. A line is a sequence of records, usually points, between and including a line-start record and the next line-end. (Graphically it is represented as a polygonal line connecting points in the sequence.) Thus, of the seven records in the example above, the line has six of them.

(There are conflicts between graphics and text editing terms - a line of text versus a line drawn, current (file) position vs. current (spatial) position, "move" records in the file vs. "move" objects on a screen. Our approach draws on information-processing terms (files and records), common mathematical terms (sets and sequences), and terms from other media - "splice" is temporal while "move" is spatial. Other terms are defined as they arise.)

In the descriptions below, <text> indicates some text string, <pt> represents some point entered from the tablet, <pts> is a sequence of such points, and <record> indicates some record in the file.

A current record position, named C, is maintained by EDIGER, which can be set by commands like the following:

```
top      - position at file header.
           (just above first record)
bottom   - position at last record.

find <pt>
  - position at the record with
    point closest to this point.

find /pin22/
  - position at the next record which
    has substring "pin22".

find C+20
  - position at the record twenty
    records down from current position.
```

For displaying file contents, the user issues:

```
show
  - displays the file on an on-line
    graphics device, and/or printer, as
    appropriate.
```

The file represents textual and spatial data entered in a time sequence, and this is reflected in the modification commands. The usual insert and delete operations which add or delete records at the current location are augmented by, for example:

```
reword /pin22/pin20/
  - change substring "pin22" to "pin20"
    in the current record.

move <pt>
  - replace the point in the
    current record by the
    point entered.

splice /drill 65 header/
  - reposition current record after
    the next record in the file with
    substring "drill 65 header"
```

Other commands are similar: copy is a splice with no delete of the original record. Rotate and scale are spatial operations which can be applied to individual records, but are more often applied to groups of records.

Grouping Records

Grouping is a facility for indicating subsets of records in the file on the basis of various criteria: each of the modification commands can be prefixed by a subset former with an easily anticipated result. For example, commands are given below which say things like "translate everything inside this polygon this amount", or "delete all records which have substring 'pin20'", or "attach to each point near this point the label 'drill65', and splice them at this location in the file".

The current grouping criteria are:

```
substring /pin20/
  - the set of records with the
    substring "pin20" (we will use the
    term "match the substring").

between <pt> <pt>
  - those records between the
    records containing the points
    indicated.

line <pt>
  - the line containing this point.
```

```
inside <pts>
  - those records whose points are
    inside the polygon formed by
    <pts>.

near <pt>
  - those records with points within a
    preset distance of this point.
```

Some examples of use:

```
inside<pts> move <pt> <pt>
  - all points which fall inside
    the polygon defined by <pts>
    are spatially translated.

substring/pin20/ delete
  - delete all records which match
    "pin20" (i.e. contain this string)

near <pt>  insert/.lb drill 65/
  - attach label "drill 65" to records
    with points near the point.

CS splice /drill 65 header/
  - splice the "current set" to a new
    location (see below).
```

Current Set

Each operation on a single record leaves C in a well defined location; for example, after "delete", C is at the record above the record deleted; after "insert", at the record most recently inserted. After a group command, the "current set", or CS, is the set of those records at which C would have been left had the command operated individually on each record; CS is thus a natural extension of C.

The command in the last example splices the current set, those which in the previous line have had the label "drill 65" attached, after the next record in the file which matches "drill 65 header".

Set Operations

Sets formed using grouping criteria can be named for subsequent set building operations for those occasions - perhaps rare - in which some combination of criteria must be used to specify a group. Intersection, union, and difference (symbols *, +, -) operations are supported on up to nine sets.

For example:

```
substring/pin22/  set 1
inside <pts> set 2
substring/red/ set 3
```

Then

```
s1*s2 set 4
  - set "s4" is the intersection
    of sets s1 and s2.

s3*s4 reword. . .
  - modify records inside the polygon
    which match both strings "pin20"
    and "red".
```

Menus

Menu hits are macro invocations: a point entered which falls inside a menu area is subjected to a proximity test against those points in a corresponding menu file which have the label "macro". Subsequent records from the menu file, until a record labelled "macro end", are transmitted to the command interpreter.

Thus, even though most commands have one-character abbreviations for keyboard entry, menu entry is more convenient. Menu files are created like other EDIGER files, being treated as menus only after the user so designates via an appropriate command.

The Environment

The user divides the tablet into one or more rectangular sub-areas, and specifies coordinate reference information and associated file for each. Several sub-areas can map to the same file, and several files can be defined, though only one, the "subject file", can be open for modification at any one time; the others are menu files supporting the description and modification of the subject file. Transformations from device to reference coordinates, and expansions of menu hits, are performed before command interpretation.

There are commands for loading and saving files on disk, and for storing environment information for later use in another task.

Implementation

EDIGER is device-independant; the "tablet" above could be the screen of a graphics terminal, for example; only very basic device driver for each is required. The system is written in Fortran, with several Assembly programs for interrupt handling and dynamic file definition. The editor holds the file "in-core", using for list and text storage very large arrays (several megabytes) which fortunately incurs little overhead on our demand-paging operating system (TSS).

One of the many useful lessons of Software Tools [1] concerns the concept of filters, which (p.64) encourage "a standard representation for text to be passed between programs or to be stored on files for later use". Ideally, the input language of a program should be flexible and its output readable by subsequent processors. The files produced by EDIGER are very simple, primarily due to the minimal set of primitives it supports.

Applications

The user with the PC artwork needed to enter coordinate data, together with some additional text information to indicate drill changes and a shorthand entry of some typical point patterns - double rows of equally spaced points, eight or ten or more per row ("DIPs"), and long strings of equally spaced points. Two menus were defined, one for editor commands, a second for generating insertion text representing:

1. a drill-size change,
2. a DIP pattern, or
3. an equidistant point sequence.

The user records points in the sequence which they are to be drilled, choosing drill changes from the menu as necessary. DIPs are indicated by attaching labels (via the menu item) to two determining points on the DIP pattern, i.e., the minimal information necessary to identify the pattern. General sequences of equidistant points are indicated by an appropriate label on three points of the sequence (first, second, and last). It is important to note that these patterns are not expanded by EDIGER, but by a subsequently invoked PL/I program which also generates the NC drill tape from the EDIGER output file.

The second application is the solar-energy simulation problem. The user draws system components as closed lines on the tablet, with LS labelled with a component identifier. ("closed" implies spatially identical endpoints.) Then he enters its parameters (e.g., heat storage tank capacity, fluid specific heat and initial temperature) as free text. He now indicates information flow by connecting with lines the outputs of each component to the inputs of other components, distinguishing among multiple outputs or inputs by attaching labels to LS and LE indicators. After saving the file, he runs a Fortran program which scans the file and "verbalizes" its topology for the simulation package (TRNSYS) as follows: first, closed and nearly closed lines are found and interpreted as components, and open lines as connectors. A containment test (connector endpoints in component polygons see [2]), serves to identify the source of each input for each component (we assume the user drew connectors in the flow direction). Finally, a file is written in which each component identifier is followed by its parameters and its sources of input, as required by TRNSYS. This approach is similar to that of Fraser[3], who describes a circuit design application on Unix.

Verbalization filters

The human brain is divided into two hemispheres, one adept verbally, the other spatially. The communication path between the two is a structure called the corpus callosum, which is in a sense a bridge between the spatial and verbal.

The verbalization process described in the TRNSYS application illustrates a similar bridge, from words and drawings to words alone. Extensions to express things like components within components, labelled-point containment and connectivity, unlabelled points as references to previous components, etc., would result in a more general translator from action to description, possibly powerful enough to produce APT input from an EDIGER file.

The inspiring ACTION package[4], which runs on a PDP-11/55, represents an approach at a different level to the verbal-nonverbal transformation. An ACTION user is "wired" directly to a running application across a

dynamic two-way graphics interface which can be defined virtually independantly from the application. Via shared memory with the application, ACTION links the spatial senses of the user with the verbal senses of the application designer.

Application-independant efforts like these might comfort the application designer, who is loath to accept a "picture" as program input if he can get its thousand-word description. It is interesting, however, that his reaction to interaction and data preparation is again different: he tries to eliminate them, as much as possible, via powerful languages and data bases. He wants, for example, to allow an engineer to say "subject to these structural and aerodynamic constraints find me a variation of this stock airfoil which performs best over this weighted set of airspeeds." For the engineering disciplines especially, the big bottleneck for goal expression seems to be processing power.

Related Work

EDIGER was designed, basically, to capture the words and sketches of a user as he describes some process, problem, object, etc. It uses and extends ideas found in four existing editors - two graphics, two text - of which none allows free mixture of text and graphics.

The command syntax is from "edit" in [1], which is itself rooted in QED[5]. Ideas on file handling and structures were borrowed from our TSS Editor. The basic framework for graphic input - points, lines, labels - were incorporated in TABEDIT, an editor written by a student, Rosanna Lee, at the Computation Centre during the summer of 1980.

EDIGER (and TABEDIT) are probably closest in approach to the graphics editor described by Fraser[3] running under Unix for aiding circuit design, and could be considered a generalization allowing free text and more powerful editing capability.

The addition of sets and set operations was suggested by their use in "NRC Information System", currently being developed by R.A. Green, based on his earlier CAN/OLE design.

Enhancements

Both the EDIGER operations and its file formats are simple, which leads to the speculation that EDIGER could be micro-based, with a file transfer capability to the deeper software and centralized peripherals of a mainframe.

Grouping criteria might be extended. While their current orthogonality has the benefit of clarity, and while set operations provide one way to mix criteria, there are natural groups which cannot be specified using this approach. Two candidates for inclusion are "brackets", and "chains", both indicating unbroken sequences of records:

```
bracket /begin/,/end/   delete
```

would delete the sequence of records containing the current record, the first matching "begin", the last matching "end", with no interior record matching either.

```
chain <pt> move <pt> <pt>
```

would move that sequence of points in the file containing an indicated point which satisfy the condition that the maximum distance between adjacent points in the sequence is no more than a pre-set tolerance. Among other things, this pattern might be used to indicate dashed lines and sequences of lines which draw text.

Naturalness

"Every program defines an input language, albeit a primitive one" [6]. If we accept for the moment the primitive nature of this language, one compelling measure of its naturalness lies in comparing communication using it against human-human communication of the same information.

This paper began with a description of everyday man-to-man communications involving graphics as a starting point for developing a more natural man-computer interface. One should note that processors like APT and TRNSYS, are natural in the sense that the sequence and content of problem expression are close to the way people might communicate similar information over a telephone.

EDIGER however permits the user to draw as he talks, gives him a blackboard as it were, and "watches" and "listens" as the user describes his problem. Anywhere a user hunts for a scrap of paper to describe his problem - flow diagrams, circuits, graphs, etc. - seems a potential candidate for its application.

Conclusion

The "man-computer" interface is often a variation of the "man-man" interface: the user "talks" to the application designer in a language chosen by the latter. The computer, however, constrains this language to be primitive, and its interpretation literal: hence the need for a tool, an editor, which allows the user to create and revise his descriptions if they are complex.

EDIGER is a tool for recording and editing these descriptions when a mixture of text and graphics is required.

Acknowledgements

My thanks to Rosanna Lee for her TABEDIT work, to my colleagues Andy Haycock and Art Green for their help; and to Marceli Wein and Ken Steele of the NRC Graphics section for their encouragement. The last three people read a draft of this paper, and made valuable suggestions for improvement.

Bibliography

[1] B.W.Kernighan and P.J.Plauger, Software Tools, Addison-Wesley, Reading, Mass., 1976.

[2] B.K.Aldred, "Points in Polygon Algorithms", IBM(UK) Scientific Centre Report UKSC-0025, April 1972.

[3] A.G.Fraser, "Unix Time Sharing System: Circuit Design Aids", Bell System Technical J., Vol.57, No.6, July-August 1978, pp.2233-49.

[4] P.P.Tanner and K.B.Evans, "ACTION - A Graphics Aid to Interacting with Models and Simulations", Proc. Sixth Man-Computer Communications Conference, 1979, pp.49-61.

[5] L.P.Deutsch and B.W.Lampson, "An Online Editor", Comm. ACM, Vol.10, No.12, Dec.1967, pp.793-799.

[6] S.C.Johnson, "Language Development Tools on the Unix System", Computer, Vol.13, No.8, August 1980, pp.16-24.