Dynamic Screens and Static Paper

SANDRA BAISSAC SMITH

Skidmore, Owings & Merril
Chicago, Illinois

DOCUMENTATION FOR GRAPHIC SYSTEMS

Design issues for written and on-line documentation have become the subject of articles, newsletters, conferences, and seminars. Their importance in supporting and marketing software systems is no longer questioned.

For the most part, this extensive discussion has been based on structured systems: data base or data query, text editors or operating systems. (1) In such systems, user goals can be clearly identified for any one task. There is a single procedural path to accomplish the task and achieve the goal. Efficiency, and thereby documentation effectiveness, are measured in terms of the time necessary to learn the path, and the number of errors encountered.

Graphic building design systems in an architectural practice serve a dual role as an efficient means of drafting production, and as a visual tool within the creative design process. These systems must allow the same freedom as the traditional pencil and yellow tracing paper. They may be conceived and designed as unstructured systems on several levels.

Unstructured Input
Unstructured Paths
Unstructured Tasks
Unstructured Data

Each is a source of potential ambiguity, and each affects the documentation effort.

This paper will discuss the types of problems encountered in creating documentation for such unstructured programs. Questions will be raised as to the approach to printed and on-line reference documents, as well as training. Examples of tasks and documentation will be based on the graphics programs developed by Skidmore, Owings & Merrill (SOM) for the use of its professional staff. These programs process two- and three-dimensional graphic data, interface with plotting devices and with engineering analysis programs.

1. Introduction

Graphics systems, processors and display units have evolved rapidly over the past five years. Input devices, once limited to alphanumeric keyboards, now include a wide variety of interactive devices such as tablets, light pens and mice. Where display screens were monochromatic, they now offer 256 different colors or more. The time needed to refresh a screen of complex data has been significantly reduced. Dialogue areas function independently of the graphic workspace, using a separate window or a separate scrolling layer of the display screen.

Building design programs have also evolved to take advantage of new technologies. However, syntax or user interface decisions based on an earlier technology, are difficult to change without affecting the productivity of trained personnel. Many of the issues raised by SOM's current programs find their solution in today's graphic capabilities. Viewing these decisions and their impact through the needs of documentation can be a profitable exercise for the programmer as well as the technical writer.

2. Unstructured Input

The building blocks of most graphics systems are geometric primitives such as nodes, lines,

polygons, text, and symbols. They are located in three-dimensional space by assigning their end points or reference points to precise XYZ coordinates. Attributes such as color, pen weight, line type, text font, etc. are attached to each element.

In architecture or engineering systems, coordinates can be entered by typing XYZ locations in real numbers at the keyboard. Coordinate data can also be defined by referring to an existing node number in the graphic data base. Both methods are cumbersome and slow; both leave ample margin for input errors. Documentation is, however, a straightforward affair.

More commonly, coordinates are entered interactively by placing a graphic cursor at the desired location, and indicating actions with either a key stroke or a combination of mouse buttons. Examples in written documentation must use graphics to portray these choices effectively.

Although the use of these devices is enjoying considerable vogue, the command input entails two notions that do not lend themselves to documentation. Where screen refresh time is a program constraint, input is often imperfectly echoed. That is to say, the visual display may not clearly indicate which button or key has been pressed.

Secondly, input depends on a spatial and temporal sequence that may be quite complex. Consider the technique SOM uses to add a rectangle of structural beams onto an underlying graph paper grid.

The verbal explanation as written follows:

> Use the S key to select points off the graphic mesh and <ESC> to interrupt the continuous sequence if necessary. The X key will find the existing temporary node at the I node of the first beam and use it as the J node of the last beam element. Press E to enter the beam list.

The illustration as printed is:

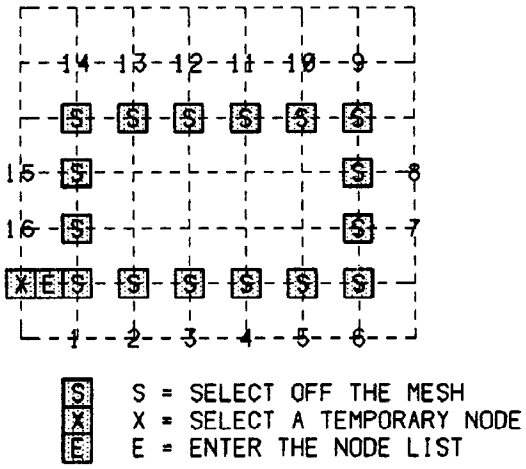


FIGURE 1

Figure 1 attempts to minimize one inconvenience of graphic input. A symbol appears where the key would be pressed. The spatial movement of cross hairs is noticeably absent. The notion of temporal sequence, although implied in the numbered positions, is entirely lacking from the X E S key sequence. A full set of illustrations would have to include 17 separate images, indicating each spatial position of the cross hairs and the key used at that point in time. A more economical and effective set might include the first position and key, then the last position and keys.

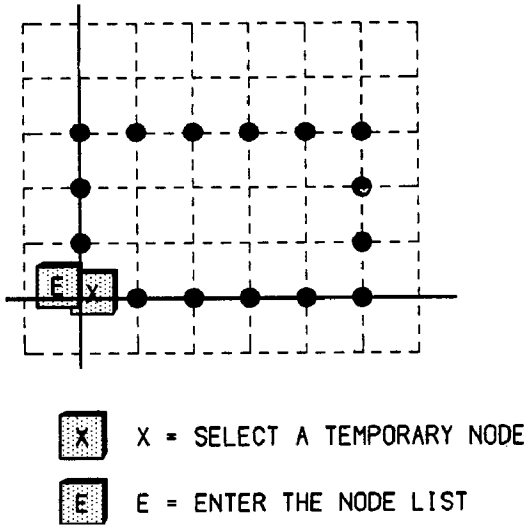


FIGURE 2

A similar problem arises when input is divided into subsets. Where text subsets of sentences and paragraphs may be indicated by a space or a carriage return, a series of disconnected lines or discrete polygons needs a more complex convention. SOM systems use the escape key as the equivalent of picking up the pencil.

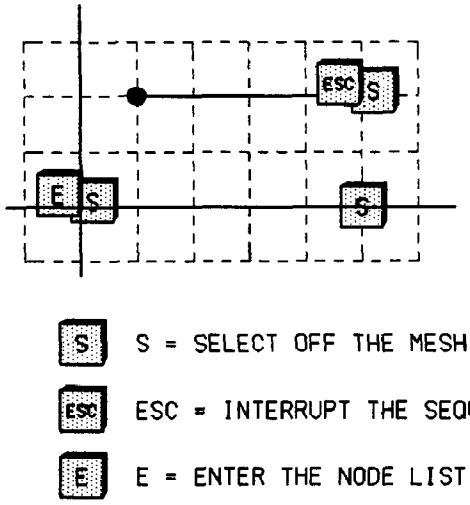


FIGURE 3

Non-echoed input, be it through a mouse, tablet or function key, also affects tutorial preparation. Although recent studies seem to indicate that effective tutorials can screen out input errors, they have been based upon text editing command acquisition.(2) The discrepancy between the new user's expectation and the tutorial behavior is far greater with spatial representations. Figure 4 shows what a novice might see if he placed the cross hairs incorrectly to enter a new node in such a tutorial.

```
DRAF> SET MESH X Ø TO 7 Y Ø TO 4
NOW ADD A NODE GRAPHICALLY.
USE THE S KEY TO PICK OFF THE MESH.
PICK THE INTERSECTION X=4. Y=1.
DRAF> ADD NODE .
```

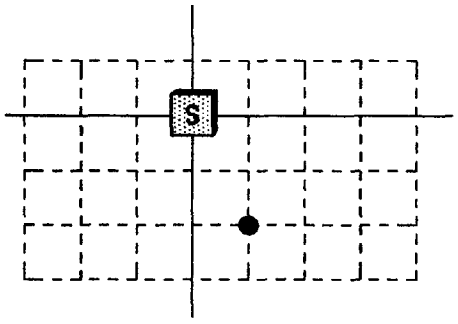


FIGURE 4

Command language can also be considered as unstructured input, whether it be through menu choices or direct entry. If we first define a rectangle then its color fill, or first choose the fill color then define the rectangle, the final result and, in many cases, the efficiency are the same. For other commands, efficiency may be greatly affected. Consider the two commands that follow:

```
DELETE LINES IN . VECTOR 2 COLOR 4
DELETE LINES COLOR 4 VECTOR 2 IN .
```

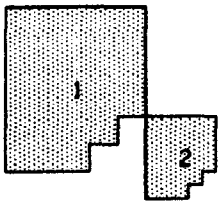
The results of both commands are identical. Those lines that are within a graphically defined area, are blue and are in a dash line vector will be deleted from the working memory. However, the first formulation may mean that the computer works through a shorter list of blue dash lines. How does one convey that notion to a group of novices who will be sharing the same computer resource?

Unstructured input language may, in other cases, have varying results according to the order in which options are used. SOM systems, developed when dialogue areas often overwrote the graphic workspace, allow for a maximum of operations within a single command. The resulting syntax is difficult to map in a reference form. Existing documentation conventions precede additive lists with an ampersand & and enclose optional choices in parentheses **()**. Where the user must supply data, the data type is enclosed between left and right carets **<>**.

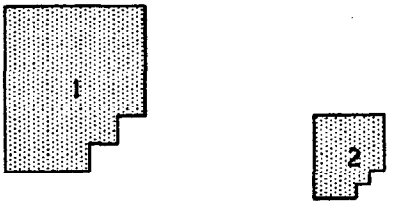
A partial syntax diagram listing transformation options might read:

```
ROtate <x angle> (ABout <list xyz>)
       &<y angle>
       &<z angle>
        X <angle>
       &Y <angle>
       &Z <angle>
RX <angle>
&RY <angle>
&RZ <angle>
&SCale <scale x> (ABout <list xyz>)
      &<scale y>
      &<scale z>
       X <scale>
      &Y <scale>
      &Z <scale>
SX <scale>   (ABout <list xyz>)
&SY <scale>
&SZ <scale>
```

No hierarchy is or can be implied. However, the order in which data is translated, scaled and rotated is crucial. Figure 5 illustrates the results of repeating a polygon with both translation and scale options.



DRAF> REPEAT POLY 1 TX 3 SCALE .5



DRAF> REPEAT POLY 1 SCALE .5 TX 3

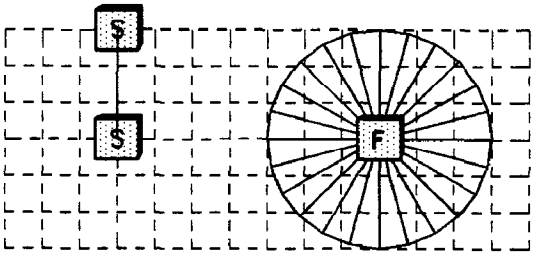
FIGURE 5

3. Unstructured Paths

The methods of building a graphic image: positioning, scaling, coloring, etc. are as diverse as the individual user's approach to design and the creative process. There is no one defined path to accomplish a task. Indeed, there is often no one defined sequence for options within a single command. Efficiency cannot be determined by task accomplishment, nor can the effectiveness of training or documentation.

In training and in use, there is no difference between the following two paths to the same geometric construct.

DRAF> ADD LIN .
DRAF> REPEAT LINE RZ 15 ABOUT . COPY 24 LINK



DRAF> ADD LINE CIRCLE 24 RADIUS .
DRAF> REPEAT LINE SCALE 0 ABOUT . LINK

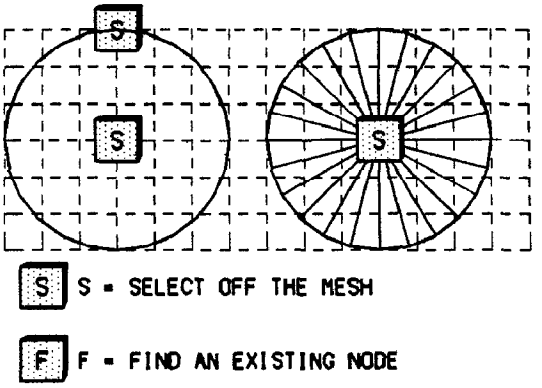


FIGURE 6

The dilemma becomes a subjective one. This flexibility can be detrimental in that it affects learning time and long-term memory.

4. Unstructured Tasks

In addition to the traditional tasks of developing working drawings and other architectural documents, tasks can include preliminary massing within a city context, site planning, development of alternative design details, presentation graphics, slides, or film animation. User Guides and other training materials must cover many of these tasks yet not overwhelm the novice user.

With data base or text editing systems, the solution to training materials can take the form of an example task. To accomplish his own goals, the student need only follow the same structure with his own contents. The structure itself is, in some ways, reassuring in its more familiar elements. Most of us recognize the final product: a table or a page of text. Many of us understand the steps to achieve the goal by transfer of existing knowledge of typewriters, files, etc.

To build examples of sufficient graphic complexity for the architect or engineer is another matter. Consider the implications of the following sample.

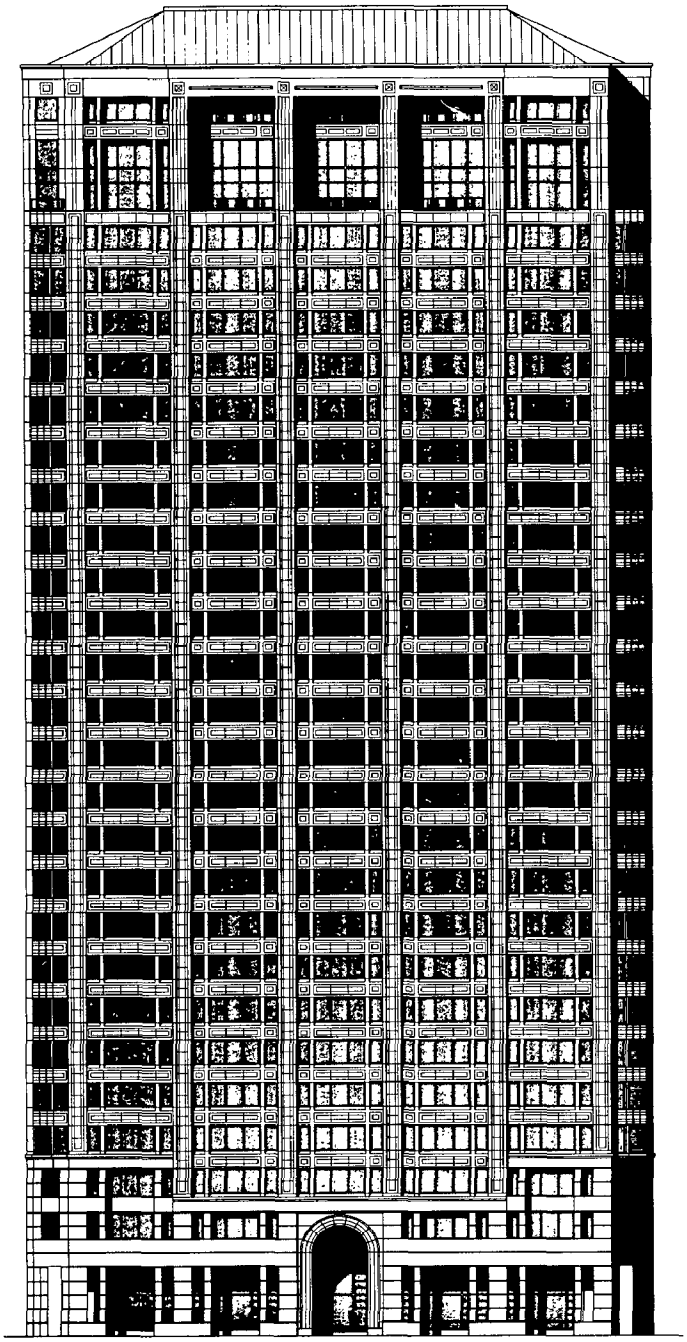


FIGURE 7

At minimum, the concepts include geometric primitives, pen, and shading; perspective, shadow, and scale; filing, and retrieving graphic data, and converting the data into a plottable form.

Despite a good grasp of both geometry and the desired architectural result, some constructs are arduous. Particular sequences may only be adequately explained with graphic aids. For example, the text:

> PAir and FIT options permit rotation and scale for a symbol instance to be specified by indicating two or more nodes between which it is to fit. The symbol definition should be 1" wide at each axis to be affected by scaling. The PAir option will scale and rotate the symbol instance so that its X axis coincides with a line projected between the two nodes. FIT 2 functions in the same manner, using the X distance to scale all three dimensions. FIT 3 allows you to define a plane on which the symbol will lie, and a Y axis scale that is the distance between points 1 and 2 and point 3. The Z scale factor will be 0. FIT 4 defines a three-dimensional location for the symbol and a Z axis scale that is the distance from points 1, 2 and 3 to point 4. Add the SCAle option to use the distance between points 1 and 2 as a uniform XYZ scaling factor.

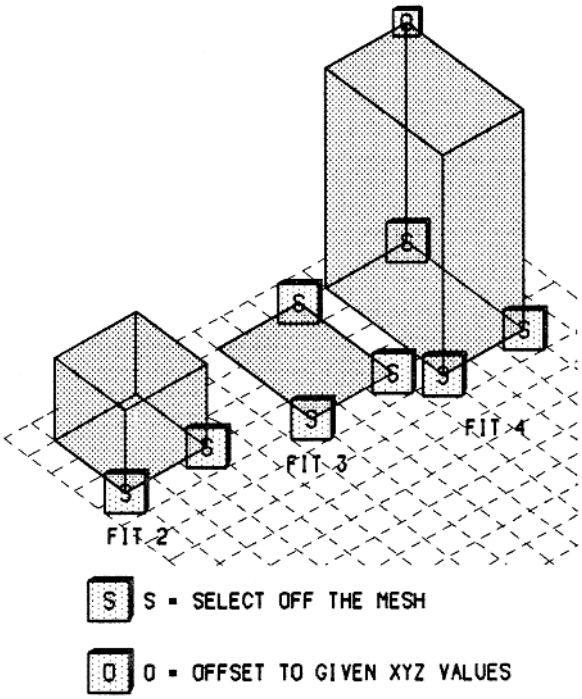


FIGURE 8

While written documentation can accommodate such simple illustrations, on-line help is limited by questions of response time, data storage space and terminal windowing or display

capabilities. The disruption factor of 80-character text scrolling across a complex graphic is high. A superimposed illustration would, for most SOM users, be entirely unacceptable.

To meet the requirements of interactive display, or even of reproduction quality, graphic examples must therefore be simple. They must also be carefully chosen to avoid any implication of a limitation to the creative process or to the architectural or engineering expertise of the computer novice.

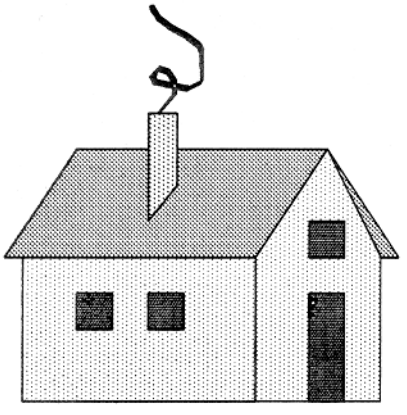


FIGURE 9

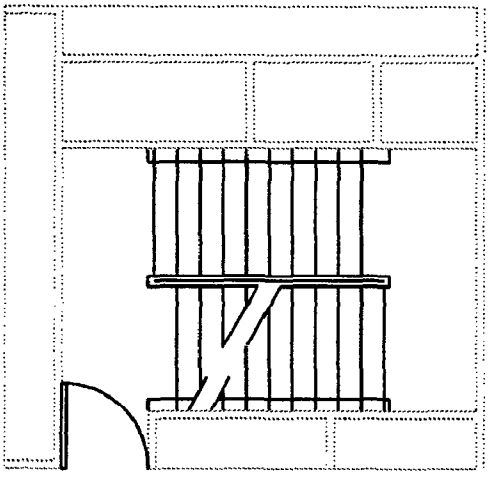
5. Unstructured Data Management

Management of graphic data, both during work sessions and in permanent memory, can allow the user complete liberty. In SOM systems, up to 63 "layers" or overlays may be used in a single work session as temporary divisions of the graphic image. Nine group numbers provide internal structure tags within the graphic data. A single image is often broken up and stored in an large number of graphic files. For many, these are not intuitive concepts.

When files are loaded into the terminal display memory for manipulation, no display feature, counter or log automatically remains as a visual indication of these divisions. While the display can be changed to indicate layers or groups by color coding, most designers prefer to develop and view an image in the colors they have chosen for the final color plot, slide or client presentation.

Thus, any graphic or verbal representation of these divisions is, at best, artificial. It may also be lead to misconceptions about the graphic display or internal structure of the data. For example, setting a color mode to reflect the temporary layer divisions will change the color attributes of the data. However, the next time that data is loaded into the working memory, the colors will not determine any distribution into layers.

In the same way, printed illustrations of layer divisions may reinforce a false notion of permanence, or of shading value. Figure 10 below was produced using the halftone capability of an electrostatic plotter. Plotting the data from temporary layer 2 will not automatically result in a halftone plot.



DRAF> PLOT LAYER 1

DRAF> PLOT LAYER 2

FIGURE 10

6. Summary

Documentation for graphic systems calls for a thoughtful and disciplined approach to the use of illustration and text. The information to be conveyed and the skills to be acquired are visual in nature. The translation is one from a dynamic screen to a static paper or on-line display.

Beyond issues or solutions for the documentation writer, the dilemmas outlined in this paper pose three challenges.

From human factors research we need a better understanding of the way we acquire underlying skills or measure efficiency for graphic tasks. Error blocking in tutorial models could be extended to explore spatial anticipation.

From software designers, we need effective ways to display and manage graphic examples in on-line materials. Dual-processing and windowing offer interesting possibilities that need to be evaluated in terms of their cost in overall system performance, network performance, and display response time.

From application designers, we need a constant dialogue and cooperation in circumventing those display or syntax decisions that may make a powerful creative tool difficult to teach and to learn.

References

The statements herein are the result of general readings from Communications of the ACM, ACM Computing Surveys as well as the newsletters of SIGDOC, ACM special interest group on documentation, and SIGCHI, ACM special interest group on computer - human interaction. Other publications that draw their examples primarily from text or data entry systems include FOLIO, Sandra Pakin & Associates.

Special reference is made to:

Carroll, John M. and Dana S. Kay, "Prompting, Feedback and Error Correction in the Design of a Scenario Machine", Proceedings, CHI '85, Human Factors in Computing Systems.