ANALYSIS OF A 3D DESIGN LANGUAGE

David P. Makris
International Business Machines Corporation
P. O. Box 390, Poughkeepsie, New York 12602

ABSTRACT

This paper presents an analysis of an interactive system and attempts to improve the human factors using methods derived from recent publications. The technique used to do this is more valuable than the resultant interface, since it could be a tool for other developers trying to build user-friendly systems. It may be used before any code is written to find usability problems early in the design process. A working solid-modeling system will be described, and then a new "action language" will be proposed on the basis of a more concise set of interaction rules aimed at improving its usability. While not going into the detail of an exhaustive formal grammar description, a similar method is used on representative sequences to minimize the action language. The "best" means of designing in three dimensions is beyond being described in this short paper. For this reason the topic will be constrained to one particular version of the application running on one hardware configuration.

KEYWORDS: Interactive Graphics, Human Factors, Interface Language.

INTRODUCTION

The GDP (Geometric Design Processor) modeling package was developed in a research environment and was originally based on a keyboard command language to which menu capability had been added. A detailed analysis of some sample command sequences will be used as a basis for constructing a new "action language" that makes better use of the input hardware and requires fewer user operations for each interaction. The "rules" will also be analyzed to provide a consistent interface of lesser complexity and with fewer possible sequences of input actions. This should improve the human factors by making it easier to learn. The use of input devices will also be analyzed and modified to reduce user effort required to accomplish a given set of tasks.

Numerous papers have discussed the languages of human-computer interactions, and recently these have included actions such as light pen and function key operations as language primitives. The result is commonly referred to as an "action language". By doing this, complete sequences of user inputs can then be described in a formal method that lends itself to objective analysis.

Most human factors however, are largely considered to be subjective items, and crafting a user-friendly interface requires more intuitive feel than objective design. By starting from an objective framework, the essence of the input language can be analyzed on its own merits. Hopefully the end product would benefit from this effort and lead to a better interface. This author feels that the current state of the art, however, still requires a good deal of artistry to produce a system that will be easy to use by nonprogrammers.

WHAT IS GDP?

The Geometric Design Processor was developed at the IBM Thomas J. Watson Research Center in Yorktown, New York.

It grew out of a robotics project which required software that could plan collision-free paths for the robot arm in a surrounding environment. The system and all of its commands are described in great detail by Fitzgerald, Gracer, and Wolfe (9). The reader is referred to this paper for a complete list of functions and capabilities of GDP.

In 1975 no package was available that could model 3D objects and determine interference problems. As a result, GDP was written in-house to fill this need, and has since been enhanced so as to be usable as a general purpose solid modeler. Over the years new functions and better interfaces have been added with the help of mechanical designers who used the program on a limited, experimental basis. They have studied the user's viewpoint and made suggestions which have made GDP more understandable for the noncomputer professional. The input of many diverse users has made it very robust and flexible. The implementation also has many features added to enhance its extensibility, resulting in a system more powerful than any commercially available.

A useful tool for the mechanical designer skilled in Computer Aided Drafting, I believe that it now requires a simpler human interface to become a better tool for the average designer. I will analyze short sequences of commands from the interface for complexity and consistency, and show how improvements can be made using an objective criterion. By analyzing the syntax of the input "language", I should be able to objectively make changes that will result in improved interactivity. In doing so I hope to demonstrate an application of methods, found in recent publications, for building better interfaces. The end result is not intended to be the best 3D design system possible, but an example of how improvements might be made to any interactive system.

CONCEPTUAL MODEL

The general framework of Foley and VanDam (5) is used to analyze GDP. The conceptual model contains objects, relations and operations. The solid objects are: cuboid; cylinder; hemisphere; cone; laminum; volume of revolution; 3D lines; and polyhedra of merged objects (described below). The user workspace contains any number of such objects whose relationships are maintained in a tree structure.

The operations performed on these objects include union, difference, and intersection. Several objects may be merged into a higher level node in the tree structure (union), which would then contain one polyhedron whose volume is made up from its descendant polyhedra being "glued" together. A volume may be given a negative "polarity" and in effect subtracted from a positive object through the union operation, thus producing the difference between the two. The volume of intersection can be similarly computed. Objects may also be moved and rotated to alter their positions with respect to each other.

Three other classes of operations are also provided, but they do not affect the model. There are analysis commands such as computing distances, volumes, center of gravity, etc., and viewing commands such as rotate and translate eye point. Mode-setting commands determine whether to display the wireframe or remove hidden lines, use parallel or perspective projection, etc.

SEMANTIC MODEL

The semantic model details what information is necessary for each function performed on an object. For the study presented here the object definition commands are all that we are concerned with. This requires the user to select the object type desired, and its origin, dimensions, and polarity. Polarity may be allowed to default to positive. If the dimensions are specified incorrectly, the object is not added to the user's model.

SYNTACTIC MODEL

The syntactic model details the sequence information must be presented in, or the rules by which tokens form correct sentences. For GDP these sentences are composed of commands followed by

parameters. A format for this is as follows.

sentence = <select primary option>
+ <select secondary option>
+ <option dependent parms>.

This format will be followed by the action language statements given later on in this paper. In a formal grammar production the rules would be equivalenced to sentence elements, but for brevity this paper will only deal with the right hand side of the equation.

LEXICAL MODEL

The lexical model maps the language tokens onto available hardware devices. A "select" token would be bound to the light pen or the keyboard, in the case of menu input. GDP syntactically allows both devices, but at the lexical level they are handled differently. Selecting a primary option with the light pen, for example, is a simple <light pen select> operation. Using the keyboard requires using one of two methods:

<type primary option name>+<enter>
or

<jump cursor to primary index line>
+<type primary index number>
+<enter>.

CURRENT "FACE" OF GDP

For the purposes of this paper a particular version of GDP will be studied, one that runs on the IBM 3277 Graphics Attachment (RPQ 7H0284) (11), an alphanumeric terminal with a direct view storage tube attached to it (referred to as a DVST or storage tube). Figure 1 shows a fully configured workstation consisting of a 3277 terminal, hard copy unit, plotter, tablet, storage tube, and joystick. The storage tube is used to display a projection of a model from the current viewpoint, and the alphanumeric screen handles all prompting messages, menus, and text input through a standard keyboard. A joystick is used to control cross hairs on the storage tube in order to enter locations on the screen.

Some of the characteristics of GDP are due to the fact that it has been running on several graphic devices in addition to the 3277GA, like the IBM 3250 Graphics Display System (10). The developers have chosen to keep it as device-independent as possible. Some features of the 3250 must be simulated on the 3277GA and vice versa in order for it to run as well as it does on both.

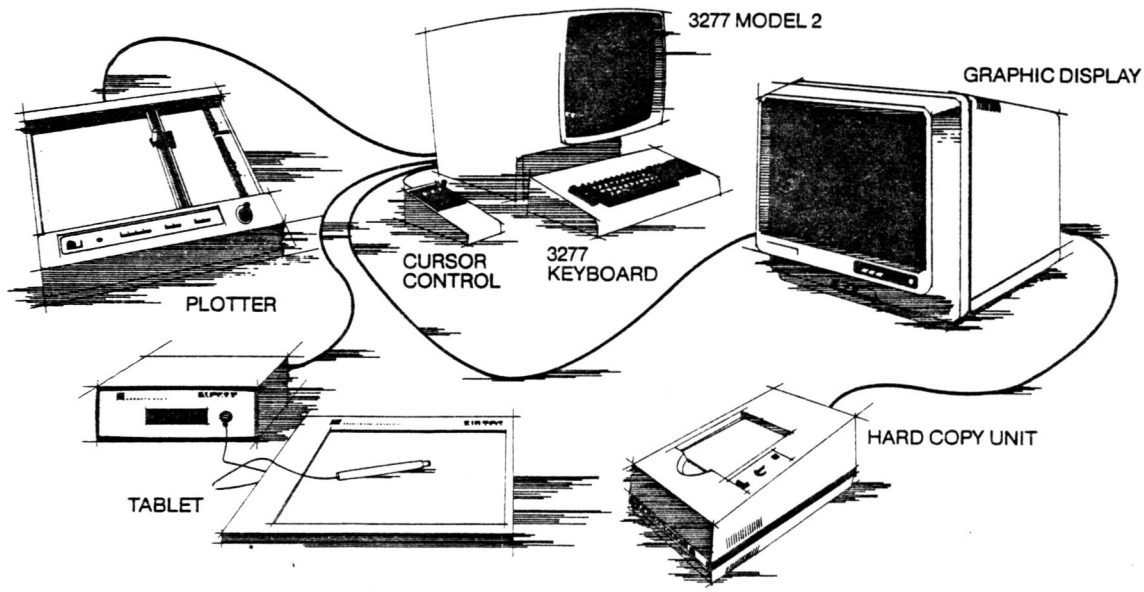

Figure 1: Expanded Dual-Screen Workstation

A recent paper by Baecker (1) argues, very effectively, I believe, that this is a wise choice but interferes with achieving the best possible utilization of a device. By its very nature this principle does not allow exploitation of the total power of any one graphics device, and is for that reason a constraint. For the purposes of this paper I will not work under this limitation; rather I will choose to risk becoming device-dependent to improve the human interface.

The alphanumeric screen is largely dedicated to a likeness of the 3250 Graphics Display System's function keyboard with the primary commands listed in boxes representing the function keys (see Figure 2). The reasoning behind this is that the average designer who will use GDP is assumed to be familiar with other 2D drafting systems which run on the IBM 3250 terminal. The average designer is expected to use the modeling system to design parts and then use a 2D drafting system to finalize the drawings used in the manufacturing process (adding notes, dimensions, etc.).

A diagram of the screen is shown in Figure 2, compressed somewhat from the original for inclusion here. The three top lines of the alphanumeric screen are for display of prompt and error messages, the bottom line for text input. Immediately above that is a

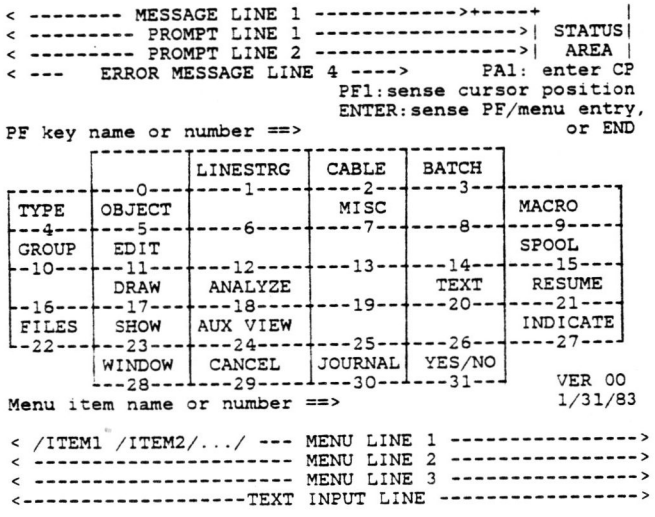

Figure 2: 3277 GA Screen Layout

secondary option menu, labeled as menu lines 1, 2, and 3 in the diagram. The input area above the menu, after the "==>" symbol, is a field for entering a secondary option index number. Lastly, above the image of the 32 function keys, a field for entering a function key index or name, again after the "==>" symbol.

USE OF GDP

A user would normally select a primary command from the 15 or so displayed in the center of the alphanumeric screen by entering the name of that command, entering the option number in the first input line, or selecting the command by light pen if the light pen option is installed on the user's device. A secondary menu is then displayed near the bottom of the screen which can be selected as above: typing the option name; entering the index on the second input line; or light-pen selecting the menu item itself. The JUMP key causes the cursor to move immediately to the next input area, so switching between the three requires no more than two keystrokes.

To enter a primitive cube, the user first selects OBJECT on the primary menu and "CB" on the secondary. A sequence of menus appears and asks for a starting X, Y, Z coordinate and the cube's height, width and depth. The cube appears in its assigned position and a secondary menu allows the user to cancel it, change its polarity (solid to hole), or, in the case of a curved object, change the faceting (roughness of the polyhedral approximation of curved surfaces).

The user may continue adding new objects, or else use the edit or analysis functions at any time. The primary option menu is always active and can be used to terminate a sub-option without completing the operation, in case a user makes an error or changes his mind.

The model is displayed on the storage tube by projecting it along a view vector controlled by the user. It may be rotated in any direction at any time. Axes are displayed in the upper right corner with the positive X, Y and Z sides labeled to help the user understand the viewing angle. The

projection may also be zoomed in on any portion for closer inspection. As objects are added to the model they immediately become visible on the DVST.

ANALYSIS

At first look, a GDP user sees the alphanumeric screen's list of groups of commands, and must choose the one that contains the function that he wishes to use. The group name is keyed in on the primary input line, the group number is typed on the secondary line, or the name is light-pen selected. This may be represented as follows:

(1a)

<key group name> + <enter>

or

(1b)

<jump cursor> + <key group number>
+ <enter>

or

(1c)

<light pen select>.

A similar action then takes place as the user selects a secondary item from the menu now displayed on the alphanumeric screen as a result of the first selection.

(2a)

<key function name> + <enter>

or

(2b)

<jump cursor> + <jump cursor>
+ <key function number> + <enter>

or

(2c)

<light pen select>.

The program would then be in the functional routine desired by the user and could prompt for any additional input. To enter a primitive cube (function "CB") the user might do the following:

(3a)

<key OBJECT> + <enter> + <key CB>
+ <enter> + <enter coordinates>.

or

(3b)

<jump cursor> + <key 5> + <enter>
+ <jump cursor> + <jump cursor>
+ <key 2> + <enter>
+ <enter coordinates>.

or

(3c)

<light pen select OBJECT>
+ <light pen select CB>
+ <enter coordinates>.

From a simple keystroke-counting point of view, an obvious improvement might be to get rid of the <jump cursor> operations by using one input area. That particular key generally requires more attention by the user than a text key, since it moves the cursor to a new screen location, maybe far from the old one. This requires that one watch the screen carefully to ensure that the cursor moves to the desired spot before typing. Rarely will a user be able to touch-type using the JUMP key and not need to glance at the screen. That extra concentration beyond what is needed to enter text makes it detract from the human factors and should be avoided. The rule in (3b) above is very rarely used by the average designer since it is so cumbersome, and so should not be construed as a major flaw in the interface under examination. It is simply left from past versions that used this mode before the light pen support was available.

If the program could successfully use only one input area, then rule (3b) above would resemble (3a), only the user could type the menu index rather than a function name at his discretion. This might require longer command names or numbers, since they must then be unique to allow the application program to determine whether a primary or secondary command has been selected. This defeats the purpose of reducing keystrokes since typing command names requires more keystrokes and is error-prone.

The <light pen select> operation is by far the simplest technique for input but must be done twice to invoke a secondary command, since not all functions are displayed in the primary menu. This would not be so if the program had a menu of all the second-level commands and did away with the hierarchy. In a way this has already been done, since the second-level command names are recognized if they are typed in at group select time. For the casual user this is of no benefit, since the names are not all displayed in a menu and thus he or she would have to memorize the command earlier on.

Without the hierarchy the user would have to be familiar with the command names and not have the benefit of grouping them into like categories. This reduces the ability of the user to find his way to a related class of operations without knowing their exact names or locations, which is something a casual user is very likely to want. For the moment this will be ignored, since a solution will be given below.

Perhaps the secondary names could all be displayed in one large menu. There are on the order of 50 commands and they might be spread out 5 per line, leaving a blank line between each for readability and easy light pen selection. This would take up a large portion of the 24 x 80 alphanumeric screen but reduce the number of key strokes necessary to invoke each function. This would probably be too large a menu for the casual user, since that many options might prove overwhelming. Some form of grouping of commands into similar classes will be needed, and is proposed below.

The grammatical rule for the example of creating a cube would roughly be as follows:

<key CB> + <enter>
or
<light pen CB>.

This sequence would replace that of lines (3 a, b, and c) given above. It is obvious that the number of manual operations is reduced. In the worst case, (3b), many keystrokes are eliminated, and in the best case, (3c), using the light pen, one of the original two is removed. The number of grammatical rules necessary to describe the input operation is also reduced, which Reisner (7) has shown to point out a more friendly system in her study of human error rates and learning time.

Conversely, it may also be argued that the command hierarchy is easier for the casual user, since the number of options presented at any one time is smaller. If the placement of commands in the hierarchy is truly meaningful, then the subtasking done in the user's thought processes will make the system easier to learn. For the moment we have seen that the larger menu does reduce the number of language production rules and key strokes. Later on a technique for maintaining the structure while still using a large menu, the best of both worlds, will be presented.

In studying the current utilization of the graphic workstation hardware, we find that the storage tube is used for display of the graphic image of the model and an occasional x/y input using the joystick. This is normally used to point to a displayed object or vertex of an object. Through the 2D screen it is difficult to enter 3D coordinates by positioning crosshairs with a joystick. The alphanumeric screen handles the menus, input areas, and prompting messages. This is where most of the interaction takes place, since the alphanumeric screen may be modified without rewriting its entire contents as must be done with the storage tube for other than incremental additions.

The largest part of the alphanumeric display is the command menu, and, interestingly enough, it never changes. One might intuitively decide to place this on the storage tube since it is of such a static nature. In this implementation of GDP the alphanumeric screen was selected to allow the use of the light pen option of the 3277 for menu selection. The joystick-controlled cross hairs could be used for pointing to menu items on the DVST but this reacts much slower than the user's hand with light pen and is more difficult to aim accurately. Hand-eye coordination is necessary unless a very large pick window is used, thus reducing the number of menu options possible.

There is also a problem with excessive eye movement using a dual screen workstation like the 3277GA. Ideally the graphics device should have a very large surface area with high enough resolution to keep everything, menus, data and messages, on one screen without sacrificing clarity for abbreviations. In reality the user has two screens that he must look at to determine his current state and plan his next operation.

With the menu on the alphanumeric screen the user must glance at the DVST (to view the current model) and back to the menu on every operation. With the menu on the DVST, attention can be concentrated on the one screen for a greater number of interactions until text is required (seldom in a solid

modeling system) or numbers are required (more frequently). This would be desirable if it weren't for the slow response of the joystick.

It is possible to connect a tablet to the 3277GA in place of the joystick, which could eliminate these problems. The tablet with puck reacts just as fast as a light pen in that it is governed by the user's hand. It still requires some hand-eye coordination, but not as much as the joystick. The tablet, then, would allow us to put the command menu on the graphic screen and select items rapidly. As an added benefit the user has one input device (excluding the keyboard) rather than two (the joystick and alphanumeric light pen). This eliminates the bother of putting down the light pen to use the joystick and vice versa. One hand can rest on the tablet puck (or mouse) all of the time, allowing more rapid user response. This is referred to as a pragmatic consideration by Buxton (2), below the lexical level and impacting the user's impression of just how easy it is to use the system.

With the graphic display the command menus can be much more elaborate than on the alphanumeric screen. Icons, when meaningful, could be drawn to represent functions and objects rather than have the user read command names. It has been argued that users can recognize icons, or images, faster than command names (8), but whether this technique slows down the casual user who does not recognize an icon is not clear, and so will be avoided for this discussion. Boxes may be drawn around similar groups of commands in much the same way as the present command hierarhcy structures them. This will allow the casual user to "home in" on logically related batches of commands when unsure of exactly what is available, only the added number of interactions that currently imposes will not be needed.

A help menu item may be provided on the DVST menu that could cause the alphanumeric screen to be used to display information about the current model - a representation of the current location in the tree structure, for example. A tutorial mode could be entered while leaving the graphic display unchanged. This allows the experienced user to completely avoid the routine messages and concentrate on his work while the casual or novice user can refer to them as often as is needed.

This may introduce added programming difficulties, since messages may need to be separated into at least two classes. Important messages that must be seen every time they occur (system going down, etc.) will have to be handled in a different fashion from routine command prompts. On some devices an audible signal can be adequate for this. The tradeoff between adding program complexity for a small human factors improvement must be made by the system architects.

The added information would make the system more self-teaching and yet be physically removed from the graphics screen, "where the action is". A "where am I, what did I do, where can I go" (6) type of audit trail can be displayed as an aid to the casual user.

A side benefit of most of these potential improvements is that they move the center of the user's concentration to the graphics screen. The importance of the alphanumeric screen would be reduced, since it is no longer involved in every interaction. This would keep the user's attention on the one screen for a longer period of time and reduce the delay in scanning between the two. The casual user could refer to the alphanumeric screen for help information whenever he wanted, while leaving the state of the graphics display untouched. It would not, however, take full advantage of the capabilities of the 3277GA since the alphanumeric screen is not involved in the average user's interactions.

SUMMARY

In the analysis we looked at the structuring of sample fragments of the command language as grammatical rules. Changes were proposed to reduce the number of rules necessary to describe the language and to make it more consistent. At no point did the nature of the application influence the changes; instead the language of interaction was treated as an entity by itself.

This demonstrates our initial goal that the human interface can be objectively studied, as with grammatical analysis, and improved. I must reiterate my statement that a good deal of creativity and intuition is also used to "craft" the better systems, but perhaps this technique may be a step towards defining more formal techniques for building interfaces of higher quality without clairvoyance.

On the other hand, some hardware-dependent recommendations were made and, though this is contrary to the popular device independent strategy, I believe it is necessary for a better system. As mentioned earlier, recent papers have also taken this position and argue the point very effectively (1). Each input device has its own unique characteristics that must be taken into consideration when a study as minutely detailed as counting keystrokes is used.

There are certainly arguments in favor of a command hierarchy rather than the proposed single menu, but the grammatical analysis shows fewer actions are required to invoke each function. Boxes, icons, and other graphic items were recommended to provide the logical grouping enforced by the multilevel menu without added keystrokes. Moving the menu to the graphics screen would also serve to minimze eye movement between the two displays.

It would also be possible to make improvements similar to those recommended above on the hierarchical system. The techniques used are independent of the application. It would be interesting to compare a correspondingly improved multi-level command system with this proposal, but simply wouldn't fit in this paper.

Interfaces better than that described here can be readily found, but it was arrived at objectively. The value behind this is that it can lead to developing consistently good interfaces rather than gambling on the results. These pragmatic factors are the closest level to the user (as opposed to the syntax, conceptual, etc.) and will determine the user friendliness as far as hand and eye movements and other such mechanical factors are concerned.

REFERENCES

1. R. Baecker, "Towards an Effective Characterization of Graphical Interaction," in Methodology of Interaction, R. A. Guedj et al., Eds. North Holland: 127-147 (1980).

2. W. Buxton, "Lexical and Pragmatic Considerations of Input Structure," Computer Graphics vol. 17, no. 1 31-37 (1983).

3. P. G. Comba, "A Language For Three-Dimensional Geometry," IBM Syst. J. 7, 292-307, 1968.

4. J. D. Foley, "The Structure of Interactive Command Languages," in Methodology of Interaction, R. A. Guedj et al., Eds. North Holland: 227-234 (1980).

5. J. D. Foley and A. Van Dam, "Fundamentals of Interactive Computer Graphics," Addison Wesley, Inc., 218-242, 1982.

6. J. Nievergelt and J. Weydert, "Sites, Modes, and Trails: Telling The User of an Interactive System Where He Is, What He Can Do, And How To Get To Places," in Methodology of Interaction, R. A. Guedj et al., Eds. North Holland: 327-338 (1980).

7. P. Reisner, "Formal Grammar and Human Factors Design of an Interactive Graphics System," IEEE Trans. Software Eng., Vol SE-7, 229-240, March 1981.

8. A. Simanis, "Human Factors in Interactive Computer Graphics," Proc. 4th Man-Comput. Commun. Conf., Ottawa, Canada, 8.3-8.12, 1975.

9. W. Fitzgerald, F. Gracer, and R. Wolfe, "GRIN: Interactive Graphics for Modeling Solids," IBM J. Res. Develop. 25, 281-294, July 1981.

10. IBM, "IBM 3250 Graphics Display System Component Description," manual GA33-3037, IBM Corp.

11. IBM, "IBM 3277 Graphics Attachment RPQ 7H0284 - General Information Manual," manual GA33-3039, IBM Corp.