stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 113/79 JULI

DUTCH GRAPHICS INTERACTION STANDARDS COMMITTEE

SOME ISSUES IN INTERACTIVE PROGRAMMING REVISITED

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

1980 Mathematics subject classification: 68B99, 68K05

ACM-Computing Review-categories: 8.2

Some issues in interactive programming revisited*)

by

Dutch Graphics Interaction Standards Committee**)

ABSTRACT

During the past year a group of people involved in various ways in computer graphics in The Netherlands have held regular discussions on topics in the intersecting fields of graphics standards, input tools and interactive programs. We feel that we have developed some clarifying definitions and principles which we would like to cummunicate. Repercussions on the recently proposed Core standard would be in the area of facilities for echoing, prompting and higher level input functions.

KEY WORDS & PHRASES: *Computer graphics, graphics standards, CORE, portability, interactive programs, input devices*

*) This report will be submitted for publication elsewhere.

**) Membership of this semi-official group consists (in alphabetical order) of: Jan van den Bos, Johan Ero, Paul ten Hagen, Rens Kessener, Henk Oudshoorn, Rinus Plasmeijer, Marleen Sint, Joop van der Star, Jan Stroet, and Arthur Veen.
Correspondence to: Paul ten Hagen, Mathematical Centre, 2e Boerhaavestr. 49, 1091 AL Amsterdam, The Netherlands.

MAN-MACHINE DIALOGUE

For many people who have to work with interactive programs it is still hard to accept that their partner is not a superhuman combining the capabilities of both a human and a big computer. In fact the machine lacks many of the qualities that are essential to make the interaction as flexible, natural and effective as we have come to expect from most human-human dialogues. Nevertheless we can improve man-machine dialogues a great deal by adhering to some simple guidelines during the design of interactive programs.

Among the features we think are important and which are often missing in present interactive programs are the following:

1. The master-slave relationship should be clear. We call the user the master if he has full freedom of choice as to the next step of the dialogue; we consider him a slave when his options are severely limited. In general, we prefer the user to be the master and the machine to be the slave. However, when dialogue efficiency demands it, this relationship can be reversed, but only during short and predictable interaction sequences. Even then the user should at all times be able to regain complete control by means of an escape or interrupting mechanism. This mechanism itself should be flexible and graceful in the sense that it allows the user a choice regarding the state the dialogue should return to and the part of the past input that should be discarded (backtracking).
2. The program and the user should be in agreement about the state of the dialogue. To facilitate this the program should either reveal its state upon request or continuously display its state or state transitions. This can be accomplished through a general "status inquiry function" which should always be accessible. When the computer is the master features like menus and parameter prompting would be of great help.
3. Amount and frequency of information exchange should be dynamically adjustable. This adjustment can be automatic (e.g. depending on the user's experience with the program), as long as the user is able to overrule it. At every point the user should be able to request detailed guidance ("help" feature).
4. Feedback should be appropriate. We will elaborate on this in a later section.

PORTABILITY IN GRAPHICS

A reasonable and pragmatic definition of a "portable program" is a "program which, when ported to another installation, can be made to run on that installation with an effort which is small (say 10 to 25%) in comparison with the effort it would take to write a completely new, functionally equivalent program". When trying to port a graphics program one is faced with general problems like local dialects of the implementation language, problems following from the difference between host and target machine such as integer capacity, and problems which follow from differences in the available graphics hardware. The later problems are by far the most difficult to overcome - may the differences between different computers be considerable, those between different kinds of graphics hardware are much greater still. The characteristics of a storage tube, a refresh display and a raster display vary so widely that it is hard to write an application program which is not geared towards a specific device.

Nevertheless an attempt is made in one of the major efforts towards standardization, the so-called Core proposal[1], to define an intermediate, device independent level. A program which performs all graphical actions in terms of Core primitives, can be ported to each installation on which the Core is implemented. Whether such an intermediate level will serve its purpose critically depends on the following properties of the primitives chosen: it must be comparatively easy to implement them on a variety of installations, they must allow the use of the most powerful features of the most sophisticated hardware available, and application programmers should be satisfied with their expressive power. One has to be aware that the first two of these requirements are often in conflict: The more powerful the primitives, the harder they are to implement on other than the best equipped installations.

Although we think the Core proposal adequately provides portability and machine independence and deserves strong support, we feel that, as far as the input part is concerned, two points deserve more attention: echoing, and facilities for constructing higher level input tools from primitive ones.

ECHO PROMPT AND FEEDBACK

Interaction comprises input and output. The output has three aspects worthy of special consideration because they are specific for an interactive environment:

Echo is an immediate response to a basic and primitive input stimulus. The only function of the echo is to acknowledge that a stimulus was received. It gives no information about the state of the program beyond the fact that the program is running and input from the device is acceptable. The echo does neither indicate the initiation or completion of any action by the machine nor does it indicate correctness of the input sequence. The response time between stimulus and echo must not be noticeable to the user. This implies that in most cases the echo will have to be initiated and generated locally (e.g. provided by intelligence in the display station). Its form is dependent on the device and its content determined by the input. Echo therefore should be part of the Core package. There should be a choice in echo modes including "no echo" (if the hardware can support this).

Prompt is a signal asking the user for input. It usually indicates what kind of input is expected and what state the program is in. The content of the prompt is application dependent and it usually implies that the last requested action is completed. For these reasons the prompt cannot be generated locally. In most cases the form of the prompt would indicate which physical device(s) the user is expected to use. For instance for a logical CHOICE input the prompt could be blinking menu items on one installation and illuminating the appropriate function buttons on another one. At least the form of the prompt is therefore device dependent and thus should be part of the Core package. The application program should be able to specify the content of the prompt and have a choice in prompting modes, including none.

Feedback is all output which cannot be classified as echo or prompt but which still strongly influences the user's part of the dialogue. It greatly determines the quality of the interaction process and therefore should be fully application dependent. It should not be part of the Core package.

These three classes have fuzzy boundaries and may in fact partially overlap. It is however helpful to distinguish the echo, prompt or feedback function in particular output in an interactive program, and to design the program behaviour accordingly.

INPUT TOOLS

The Core proposes a small set of so called logical devices, input primitives described in terms of their function only, independent (at least in principle) of any physical realisation. This last claim is doubtful. The Core primitives keyboard, pick, button, locator and valuator are modeled very closely after existing physical equivalents, namely keyboard, lightpen, button, tablet or tracker ball and control dial. As a result, implementing these primitives will be trivial for an installation where the physical devices mentioned are available in the number the Core prescribes. When they are not, they must be simulated by others which are. This may lead to rather grotesque results, like simulating a pick by blinking each item on the screen in turn and having the user push a button when the right one blinks.

Faced with an awkward implementation, a graphics programmer may easily decide not to use the Core at all. Reorganising and rewriting the complete input part of a program to be ported can sometimes lead to a much better result than using the available Core implementation, especially when the target installation is much less equipped than the host. Baecker [2] illustrates this point of view with several examples, in support of his conclusion that portability of graphics programs, at least as far as the input is concerned, is impossible. If he is right, this has consequences for the organisation of graphics programs - the part where the input is handled and the data structures used by modeling functions are assembled, should be separated completely from the remainder of the program; in that way rewriting this part when porting the program to another installation is facilitated.

Such a philosophy is not consistent with the construction of higher level input functions as proposed by (among others) Van den Bos, Hopgood and Anson [3,5,6] where the structure of the whole program tends to reflect the structure of the input.

We believe though that Baecker's view is too pessimistic. Not all simulations are as clumsy as the pick simulation just mentioned. More important, it is not clear whether the main difficulty in implementing the Core on an arbitrary installation lies in the variety of graphics devices, or in the choice of primitives. The small distance between each Core logical device and a specific physical device, is in itself a source of difficulties when one tries to map such a logical device onto any other (combination of) physical device(s).

Choosing primitives of a somewhat higher level, for example the virtual devices of Wallace [8] and, surprisingly, of Baecker [2], may take away much of the problem.

There is however room for doubt whether this is the right moment to choose higher level primitives. Only recently the higher level input functions are allowing us to gain some insight in how to perform structured input, and it will take some time until experience with such higher level functions has taught what requirements should be fulfilled by higher level primitives. Furthermore, graphics hardware is still evolving rapidly and it may be difficult later on to fit in a new, as yet unknown input device. At the same time the use of some devices now causing trouble may become obsolete. One should wait with introducing higher level primitives until the graphics community has gotten used to working with higher level input functions. However, as the Core proposal stands, there are no facilities to construct these higher level input functions. We strongly recommend the inclusion of features in the Core to make at least named sequences and sets of the presently proposed input primitives possible and to provide read functions operating on such entities. This could be a stepping-stone to the more elaborate higher level input models being investigated at the moment by several people. At the same time it would provide a testbed for the idea of higher level functions.

CONCLUDING REMARKS

One of the great and not to be underestimated advantages of the Core in its present form is, that it has a fair chance to be really used by all kinds of people. In fact, this property seems to us more important than any of the specific primitives actually incorporated, and no improvement in any single primitive is worth loosing it. With the exception of the improvements and alterations suggested earlier in this report, it seems best to us at the moment to leave things as they are. In the meantime a task force should be created to look in more detail at input primitives. This task force should keep the finger at the pulse of experiences with higher level input functions and related developments, in order to propose modifications to the Core proposal reflecting the maturity of input handling which we hopefully will have reached a few years from now. As long as porting the Core to arbitrary hardware seems impossible, it seems better to take the opposite view and consider the Core as a guideline to what hardware one should buy - at least if one expects to benefit from porting programs written at other "Core friendly" installations to ones own system.

REFERENCES

1. ACM-Siggraph GSPC, First report on graphics standards, Proc. ACM-Siggraph, 1977, San Jose, Cal., USA

2. Baecker, R. Towards an effective characterization of graphical interaction, IFIPS Workgroup 5.2 (CAD) meeting, May 1979, Seillac (Paris)

3. van den Bos J. Definition and use of higher-level graphics input tools, Proc. ACM-Siggraph, 1978, Atlanta, USA, pp. 38-42

4. van den Bos J. High-level graphics input tools and their semantics, IFIPS Workgroup 5.2 (CAD) meeting, May 1979, Seillac (Paris)

5. Hopgood, F.K. and Duce, D.A. A prodcution system approach to interactive graphic program design, IFIPS workgroup 5.2 (CAD) meeting, May 1979, Seillac (Paris)

6. Anson, E.K. The semantics of graphical input, IFIPS workgroup 5.2 (CAD) meeting, May 1979, Seillac (Paris)

7. Anson, E.K. The semantics of graphical input, Proc. ACM-Siggraph, 1979, Chicago, USA

8. Wallace, V.L. The semantics of graphics input devices, Proc. ACM-Siggraph, 1976, Miami, Fla, USA