сообщения
объединенного
института
ядерных
исследований
дубна

E11-84-770

B.Naumann, M.Rudalics

A MICROPROGRAMMED PICK DEVICE

1984

1 Introduction

Device independent graphics systems like GKS [5], or the CORE [3], provide an interface between an application program and a configuration of graphical input and output devices by defining:
- a set of output primitives, which are abstractions of the basic output capabilities of a workstation, such as line drawing or curve generation, and
- a set of logical input device classes, which are abstractions of the concrete input capabilities of a workstation, like a keyboard or a light-pen.

The concept of logical input devices implies, that while an application may not control a physical device and its behaviour directly, it allows to simulate a physically not existing device via software. One logical input device, the pick device, serves for the identification of graphical entities. Usually, an implementation will realize a logical pick device by supplying a physical representation like a light-pen or a touch sensitive panel, combined with some software.

A logical input device may be used for input in different classes. One of the most popular uses of a light-pen is for realizing an interaction, where a command is selected from a command list displayed on the screen in form of a menu. In GKS this interaction has to be performed by using a "choice device". A particular interaction technique which employs a command menu may cause more than one device to be simultaneously operative in one input class: A light-pen serves for pointing at a menu item displayed on the screen, while a number or an identifier typed in via keyboard selects the same item, depending on which device the user will invoke first. Various interactions may be underway on the same device simultaneously: a light-pen may be used at the same time for selecting a command from a menu, picking an item from a list of symbols, and dragging a cursor around. Each of these interaction tasks corresponds to a different GKS input class.

Although usually any interaction will involve input and output of graphical data, picking is the only interaction which depends on graphical output which has been routed to a workstation before the interaction may start. Therefore in a first part we will consider various aspects of graphical interaction and describe GKS output and input facilities. In a second part we will describe the environment in which our pick device operates and present an algorithm which has been implemented as a microprogram and realizes the central part of a pick interaction. Finally we will comment on some alternatives to the mainstream of device independent graphics systems and the possible impact of these alternatives on the design of our pick device.

2 Interaction

We consider an interaction a process which permits a user/operator to use one or a group of input device(s) to provide an application with a value. This process may be accompanied by some sort of feedback to the user. Typically an interaction will be performed in three phases:

In an initial phase the application has to create an environment in which the interaction may come into existence. This environment usually consists of the workstation where the interaction will take place, one or a group of input devices which will take part in the interaction, and an appropriate feedback and acknowledgment technique.

When a device is enabled (accessed) the user may be informed (prompted) that the device is ready for use. During interaction the user may receive information about the state of the interaction with the help of a feedback. Various levels, see also [8], range from no feedback at all (due to the fact that the workstation is not capable of providing an appropriate feedback or because the user/operator has temporarily suppressed feedback), over simple device dependent echo (like a cursor or a crosshair to indicate a location on the screen) and alteration of display attributes (highlighting of output primitives), to feedback which is entirely controlled by the application.

When a device is disabled (released) the environment created in the initial phase has to be deleted. An interaction may be terminated by the user, when he has input a value or explicitely refused to do so, or by the application, e.g., after a timeout has occurred.

An interaction requires the participation of up to four individual tasks which correspond to the building blocks described in [6]:

(1) Input: The input task has to register user/operator interactions on the device. Usually such operations will change the internal state of the input task.

(2) Output: The output task provides feedback and acknowledgement to the user. For simple device echo this task is closely coupled to the input task, while for application controlled feedback no direct connection between the input and output task will exist.

(3) Transformation: During an interaction the type of graphical data has to be changed, or the value obtained from an input device has to be mapped or scaled before it may be passed to the application. E.g., locator input may require a transformation of impulses from a trackball to device coordinates for providing device echo, and/or the transformation of a position in device coordinates to a position in the coordinate system of the user, before the location may be passed to the application program.

(4) Control of data flow: This task controls the external (visible) state of tasks (1) to (3). While the internal state of an input task alters whenever the user operates the corresponding device, control assures that this action (temporarily) does not affect other tasks: e.g., feedback is suppressed, or transformation is postponed.

3 Output Primitives

GKS distinguishes six types of output primitives, namely POLYLINE (a set of connected lines), POLYMARKER (a set of centred symbols), TEXT (a character sequence), FILL AREA (a polygon, hollow or filled with a pattern or hatch style), CELL ARRAY (an array of pixels), and GDP (generalized drawing primitive - a category which allows to address the particular geometric capabilities of a workstation, like a curve generator). An output primitive may be "picked" by pointing at the transformed and clipped representation of the primitive on the display surface of a workstation. This representation is controlled by one or a group of attributes which determine shape, size and appearance of the primitive. Some attributes take an active part in a pick interaction:

The PICK IDENTIFIER establishes a basic naming convention for output primitives. Like other primitives' attributes the PICK IDENTIFIER is bound modally to a primitive: consecutive output primitives passed to a workstation are assigned one and the same PICK IDENTIFIER until the latter is explicitely altered by the application. The PICK IDENTIFIER is a static attribute: once assigned, a primitive retains its PICK IDENTIFIER for its lifetime. PICK IDENTIFIERS need not be distributed uniquely: the same value may be used for different consecutive groups of output primitives.

The SEGMENT NAME provides an additional naming level for output primitives. Like the PICK IDENTIFIER the SEGMENT NAME is bound modally to a primitive: when a segment is "open" all output primitives passed to a workstation are assigned as SEGMENT NAME the name of this segment until the segment is "closed". The SEGMENT NAME is a dynamic attribute: a primitive may be reassigned a new SEGMENT NAME either explicitely by renaming the segment containing the primitive, or implicitely via inserting the segment containing the primitive into another segment. SEGMENT NAMES are distributed uniquely: the same SEGMENT NAME may be used only for identifying one group of consecutively created output primitives.

The difference between the PICK IDENTIFIER and SEGMENT NAME assignment conventions is illustrated in Figure 1. The usage of integers for PICK IDENTIFIERS and SEGMENT NAMES in Figure 1 does not imply that GKS prescribes this data type to an implementation. It is the language binding, which finally associates a GKS data type with a data type available in the particular language used for the implementation.

```
  PICK_IDENTIFIER := 4     PICK_IDENTIFIER := 5
                    PICK_IDENTIFIER := 2    PICK_IDENTIFIER := 2
                    ↓                 ↓    ↓                   ↓
PICK IDENTIFIER → 4 ---------------→ 2 --→ 5 --------------→ 2 -→
time axis ------→ t1---------------→ t2--→ t3------→ t4-----→ t5-→
SEGMENT NAME -→ 1 ------------------------------→ 2 ----------→
                ↑                                 ↑
   SEGMENT_NAME := 1                  SEGMENT_NAME := 2
```

Example: An output primitive "created" in period t_1 has assigned PICK_IDENTIFIER 4 and SEGMENT_NAME 1, a primitive created in period t_4 has assigned PICK IDENTIFIER 5 and SEGMENT NAME 2.

Figure 1. PICK IDENTIFIER and SEGMENT NAME assignment.

Non-retained primitives, i.e., primitives which have been created when no segment was open, are not pickable and therefore of no further interest to us. However, a correct pick process has to assure that non-retained primitives are not picked. Only retained primitives, i.e., primitives which have been "put into a segment" before, are pickable and may be assigned segment attributes. Segment attributes which are evaluated in a pick interaction are:

VISIBILITY - a primitive in a visible segment is displayed and may be picked, a primitive in an invisible segment is not displayed and cannot be picked.

DETECTABILITY - primitives in a detectable segment may be selected by a pick device, primitives in an undetectable segment cannot.

SEGMENT PRIORITY - a primitive in a segment with higher priority has to be preferred to a primitive in a segment with lower priority, when they both overlap and are picked in the overlapping region.

These attributes are dynamic attributes and may be reassigned a new value either by using an explicit function (SET VISIBILITY, SET DETECTABILITY, SET SEGMENT PRIORITY), or implicitely, by inserting the segment containing the primitive into another segment. The visual effect of some of these functions on the display image may be postponed on a workstation by setting the deferral state appropriately. Therefore a pick interaction has to cope with the problem, that the display image does not reflect the actual state of an application: A segment may have been made invisible by the application, but the visual effect of this operation - the disappearance of the segment from the display screen - has not been yet accomplished.

4 Input Devices

GKS relies on the logical input device concept. A GKS implementation has to simulate a logical input device with the help of the physical capabilities of a (or a group of) workstation(s). For a more distinct discussion of the GKS input device concept the reader is referred to [12], and [13]. Here we will try to give a short overview of GKS input classes and operating modes before describing some pecularities of the pick device.

A logical input device may operate in one of six input classes, namely LOCATOR (to provide a position in the coordinate system of the user), STROKE (to provide a sequence of positions in the coordinate system of the user), VALUATOR (to enter a real number), CHOICE (to select a non-negative integer), PICK (to provide a status, a segment name and a pick identifier), and STRING (to provide a sequence of characters). A logical input device may be dynamically associated with any of these classes. The GKS input device concept allows many to many relationships between logical input devices and input classes to exist simultaneously: A light-pen may serve at the same time to provide input in the classes PICK and CHOICE, while a tablet and a light-pen may be used simultaneously for selecting an input value in the CHOICE device class.

The attributes of a logical input device are an operating mode, an initial value, a prompt/echo type, an echo area, an echo switch, and an (optional) data record.

A GKS input device may operate in one of three modes: In REQUEST mode an input value is obtained by suspending the execution of the application program until the user enters the value or explicitely refuses to do so. In SAMPLE mode GKS supplies an application with the actual input value without waiting for user confirmation. In EVENT mode an input value is appended on user invocation to an event queue. With the help of special functions an application may examine the entries of this queue.

During an interaction two processes may be active in addition to the application process: a measure process and a trigger process.

The measure process has to provide a measure. A measure process for a device is in existence when the device participates in an interaction. This is the case when the device is either in SAMPLE or EVENT mode, or the device is in REQUEST mode and a request for this device is pending, i.e., an input value from the device has been requested by the application. A measure for a logical pick device consists of a STATUS, a SEGMENT NAME and a PICK IDENTIFIER. The STATUS may be OK or NOPICK. If the status is OK, SEGMENT NAME and PICK IDENTIFIER observe the following rules:
- A segment corresponding to SEGMENT NAME exists, is visible and detectable.
- A segment corresponding to SEGMENT NAME is present on the workstation containing the pick device.
- A pick identifier corresponding to PICK IDENTIFIER has been

assigned to at least one output primitive contained in the segment. At least a part of this primitive is present on the display surface of the workstation containing the pick device and is not completely overlapped by primitives in a segment(s) of higher priority.

Initial values for a measure may be supplied by the application. These values have to be checked for validity as soon as the measure process starts. When the application supplied values do not provide a valid measure for the device, they have to be substituted by device dependent values.

The trigger process synchronizes the interaction. Typical triggers are the carriage return key for string input, or a light-pen tip switch for pick input. Tight synchronization between application and measure process is obtained in REQUEST mode only: Activating the trigger process - firing a trigger - indicates that the user has finished the measure process. As soon as the trigger fires the input value is passed to the application program and the measure process is deleted. In SAMPLE mode the trigger process is inactive. An input value is provided synchronously to the application only, the device is polled by the application. In EVENT mode a firing of the trigger will cause a value to be written into the event queue. This may be realized by interrupting the application program. Conceptually, an application should not notice the firing of a trigger for a device operating in EVENT mode. Measures from different devices may be appended to the event queue simultaneously when a trigger fires. In SAMPLE and EVENT mode various measures may be obtained during one interaction, as the measure process will exist until it is explicitely deleted by the application.

Prompting is issued as soon as the interaction starts, to inform the user that a device is ready for use. Echoing provides an appropriate lexical feedback to the user about the state of the measure process during the time of interaction. The echo area - a rectangular area on the display screen - may be used for displaying prompt/echo information. A special switch allows to turn echo on and off even while interaction goes on. For pick devices the following echo types are preset by GKS:

(1) Highlight the picked primitive for a short period of time.
(2) Echo the contiguous group of primitives within the segment with the same pick identifier as the picked primitive, or all primitives of the segment with the same pick identifier as the picked primitive.
(3) Echo the whole segment containing the picked primitive.

A data record may be used to supply device or implementation dependent information. A pick data record could contain indications about the size of the (virtual) view field of the pick device, i.e., the area where hit detection is allowed, or indications about the duration of echo.

Special attention has to be given, when a logical input device operates in different classes. The interaction described in the introduction of this paper which realizes a command menu, may be implemented as follows when using GKS: A pointing device like a light-pen is initialized for choice input. The name of a segment which contains the commands in the form of text primitives has to be supplied within the data record. When interaction starts, the user is prompted by displaying the segment corresponding to this name within the echo area. Pointing at a primitive within this segment will cause the pick identifier of the primitive to be mapped to an integer. This integer is interpreted by the application program as number of the selected command.

5 The Pick Device Environment

The hardware environment for our pick device is a multi-microprocessor based intelligent graphics terminal (IGT) [9], which may be characterized as a GKS workstation of type OUTIN. The IGT distributes the various tasks of a graphics pipeline (see Figure 2) among three processors:
- a monitoring processor, responsible for communication with the host computer, dynamic memory management, and global function distribution within the IGT,
- a transformation processor, responsible for coordinate transformation and clipping of output primitives, and
- a display processor, which generates the display image on a CRT and controls the input devices.

A bidirectional three-state bus connects these processors with a common memory. The memory contains the graphical data base [15], which combines the concepts of a workstation dependent segment storage (WDSS), and a display list from which the display image is refreshed. The WDSS contains the description of retained output primitives in normalized device coordinates and implements segment insertions with the help of an instantiation concept. The display list consists of items which contain the description of output primitives in device

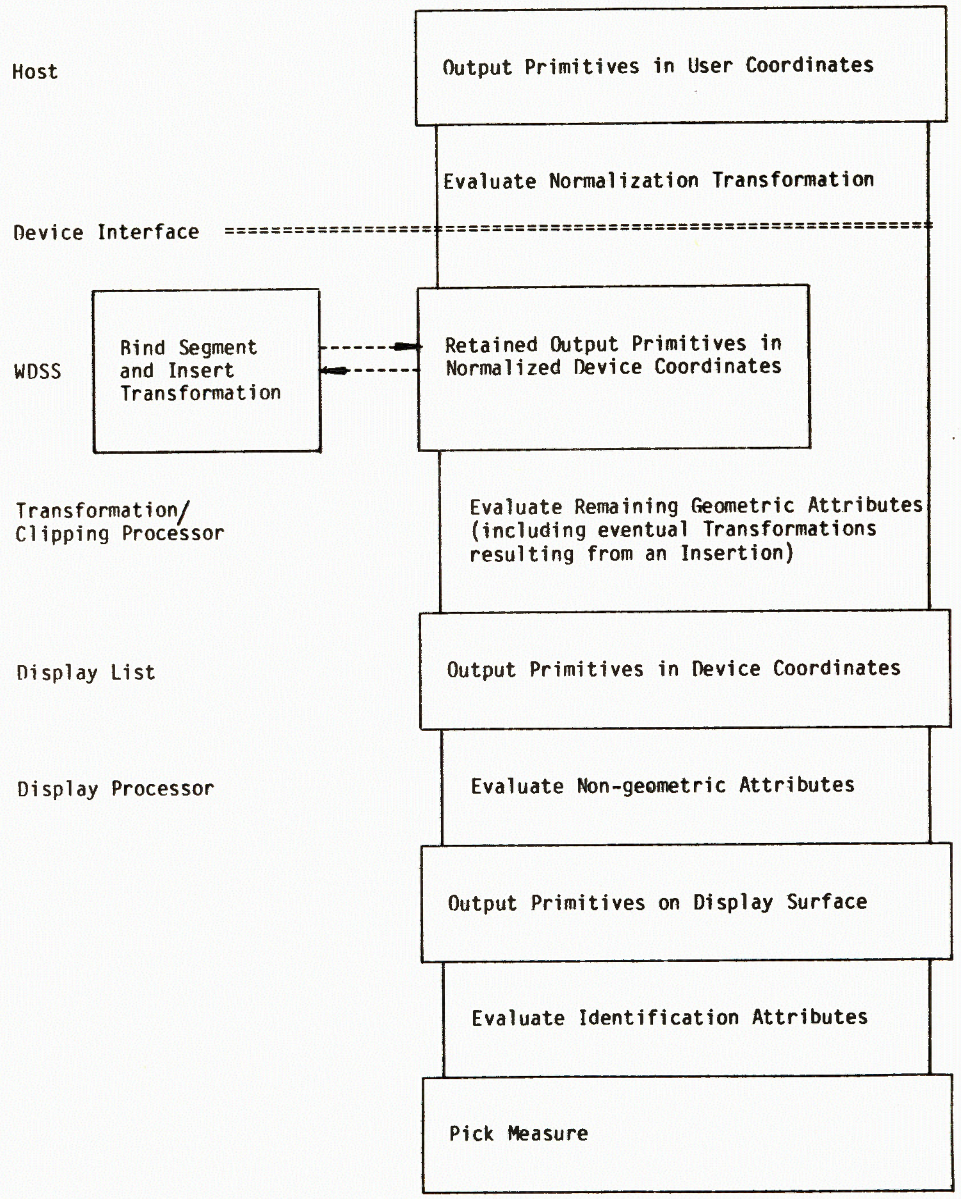
Host
Output Primitives in User Coordinates
Evaluate Normalization Transformation
Device Interface
WDSS
Bind Segment and Insert Transformation
Retained Output Primitives in Normalized Device Coordinates
Transformation/ Clipping Processor
Evaluate Remaining Geometric Attributes (including eventual Transformations resulting from an Insertion)
Display List
Output Primitives in Device Coordinates
Display Processor
Evaluate Non-geometric Attributes
Output Primitives on Display Surface
Evaluate Identification Attributes
Pick Measure

Figure 2. The IGT Pipeline.

coordinates, and corresponds to the WDSS but for the following differences:
- Primitives out of segment may be contained in the display list, the pick process has to ignore them.
- The hierarchy of graphical data is not preserved in the display list, thus no pick ambiguities may arise.

All items contained in the data base are headed by a type cell, which is used by the memory manager for identifying the type and size of the item. The display processor interprets the type header as operation code for a function to be executed, while the tail of the item is interpreted as parameters for this function. Primitives in the display list are ordered according to the priority of the associated segment. The choice segment (when existent) is the first item of the display list. Other segments are appended to the display list in falling priority. Primitives out of segment and simple device echos are contained in an undetectable virtual segment located at the end of the display list, i.e., immediately before the trap command. The trap command is necessary for synchronizing the display process with a constant time rate after a refresh cycle has been completed. Ordering segments according to their priority permit the pick algorithm to ignore priority resolution at all.

Segment names and pick identifiers are stored in the display list as integers, any translation to a different data type has to be performed on a higher level. Implicit specification techniques like pointers or indirect references have not been employed: Due to the concepts of deferred actions and EVENT mode input an item may be deleted from the display list while interaction goes on - a dangling pointer would result. Storage in the form of a text string would have defeated the capabilities of the more specialized components of the display processor.

The display processor (see Figure 3) consists of:
- a universal fixed instruction set microprocessor based part (UP), which realizes communication and synchronization with the other processors and serves various input devices like the keyboard or a trackball, and
- a special graphics processor (GP) based on bipolar processing elements. The GP autonomously scans the display list to generate output on a vector refresh tube and handles input from the light-pen. While the GP is capable of independently reading data from the common memory, it may write only to a dual-port memory situated between the UP and the GP. Therefore, any information about an interaction with the light-pen has to pass through the dual-port memory where it can be read by the UP. The light-pen

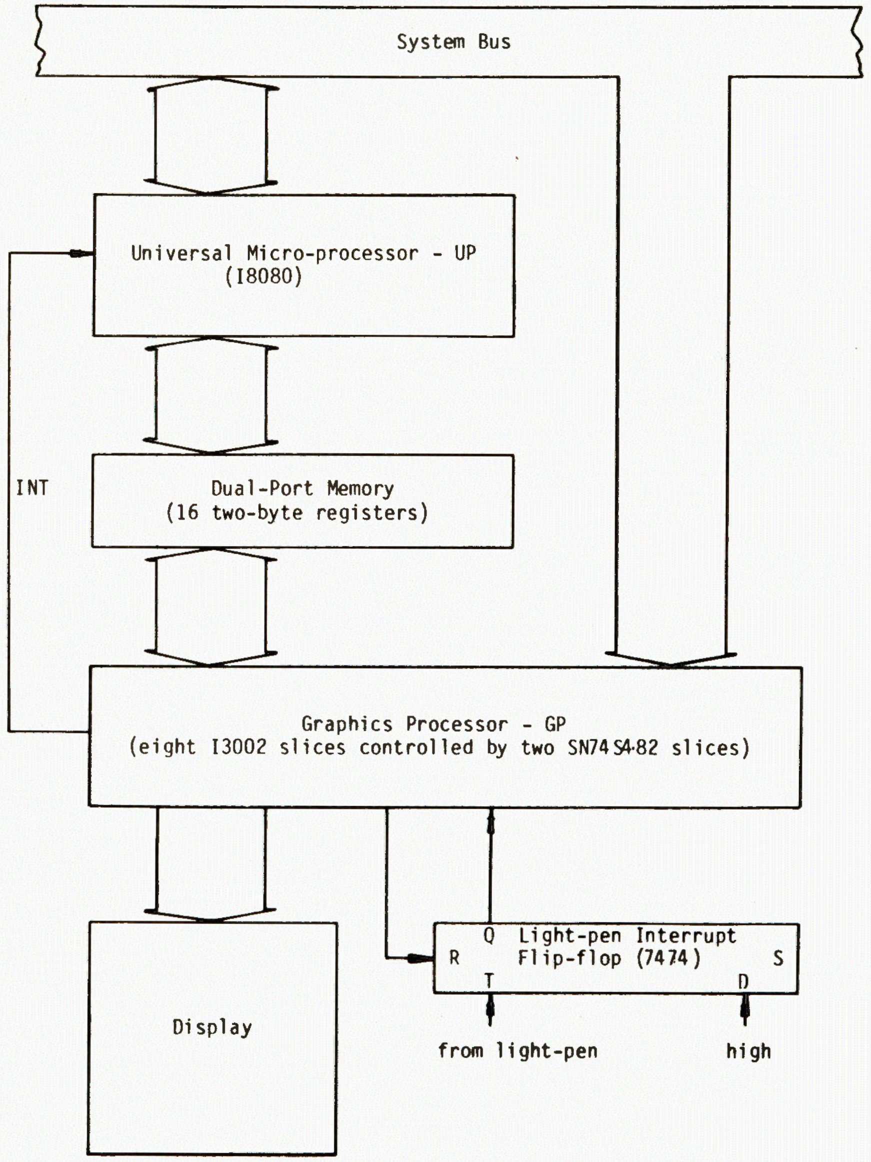
System Bus
Universal Micro-processor - UP (I8080)
INT
Dual-Port Memory (16 two-byte registers)
Graphics Processor - GP (eight I3002 slices controlled by two SN74S482 slices)
Display
Q Light-pen Interrupt Flip-flop (7474)
R
S
T
D
from light-pen
high

Figure 3. Structure of the Display Processor (simplified).

consists of a simple optical element which reacts to the passing beam of the vector device under the viewing field of the light-pen. Impulses from the light-pen are written into a light-pen interrupt flip-flop which is polled by the GP. The d-latch used for realizing this flip-flop goes high with the first impulse from the light-pen and remains high until it is explicitely cleared by the GP.

6 The Pick Algorithm

The pick algorithm has been designed according to two rules:
- Time critical sections, like identification of the picked primitive and performance of echo have been implemented as microprogram on the bit-slice processor (GP).
- Other parts of the algorithm, like interpretation and propagation of the pick measure are implemented on the universal microprocessor (UP).

In Figure 4a and 4b the interaction part of the pick algorithm is described in the form of a state transition diagram. Recently, state transition diagrams (or state graphs) have been used to describe the external behaviour of interactive systems [4], [7]. Our pick algorithm describes the internal behaviour of a pick automaton which reacts to user activation in a defined way. Each state of the automaton is represented by a circle. A transition between two states is represented by a directed arc. Transitions are defined by:
- conditions which have to be satisfied for the transition to occur (conditions are indicated by expressions enclosed within parentheses), and
- side-effects the automaton will perform when the transition is made (side-effects are indicated by assignment statements seperated by semicolons).

When a transition is made, other side-effects may occur which have no impact on the execution of the pick algorithm. The evaluation of a condition is based on the analysis of the operation code of a function the automaton has to perform. Three transitions which do not cause the automaton to change state have not been indicated in the figures as they may occur in any state of the automaton:
- When a new segment is encountered (op_code = SEGM) the name of this segment has to be remembered as current segment name (c_segm_name := segm_name) and the current number of the

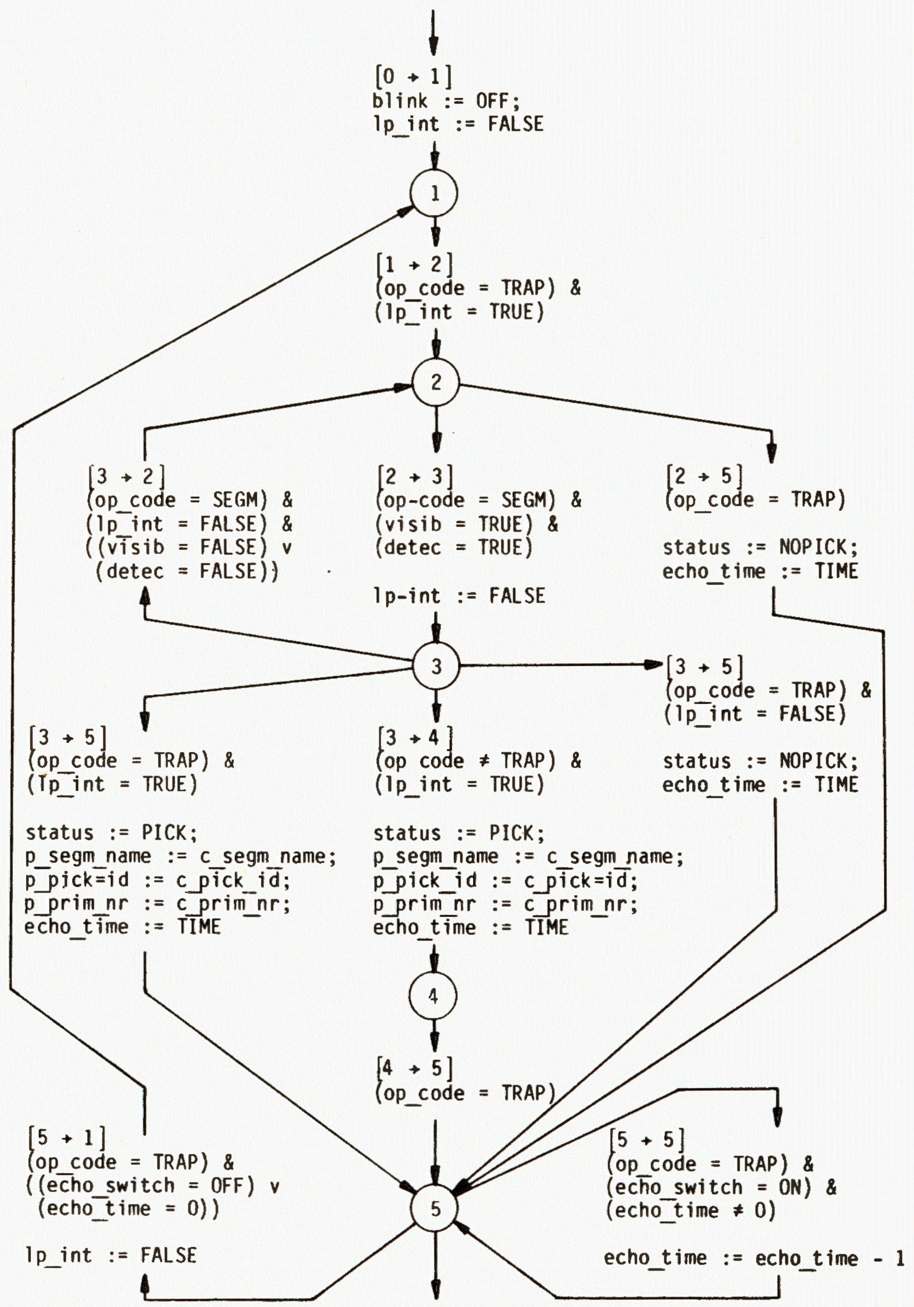

Figure 4a. Pick Algorithm - Identification.

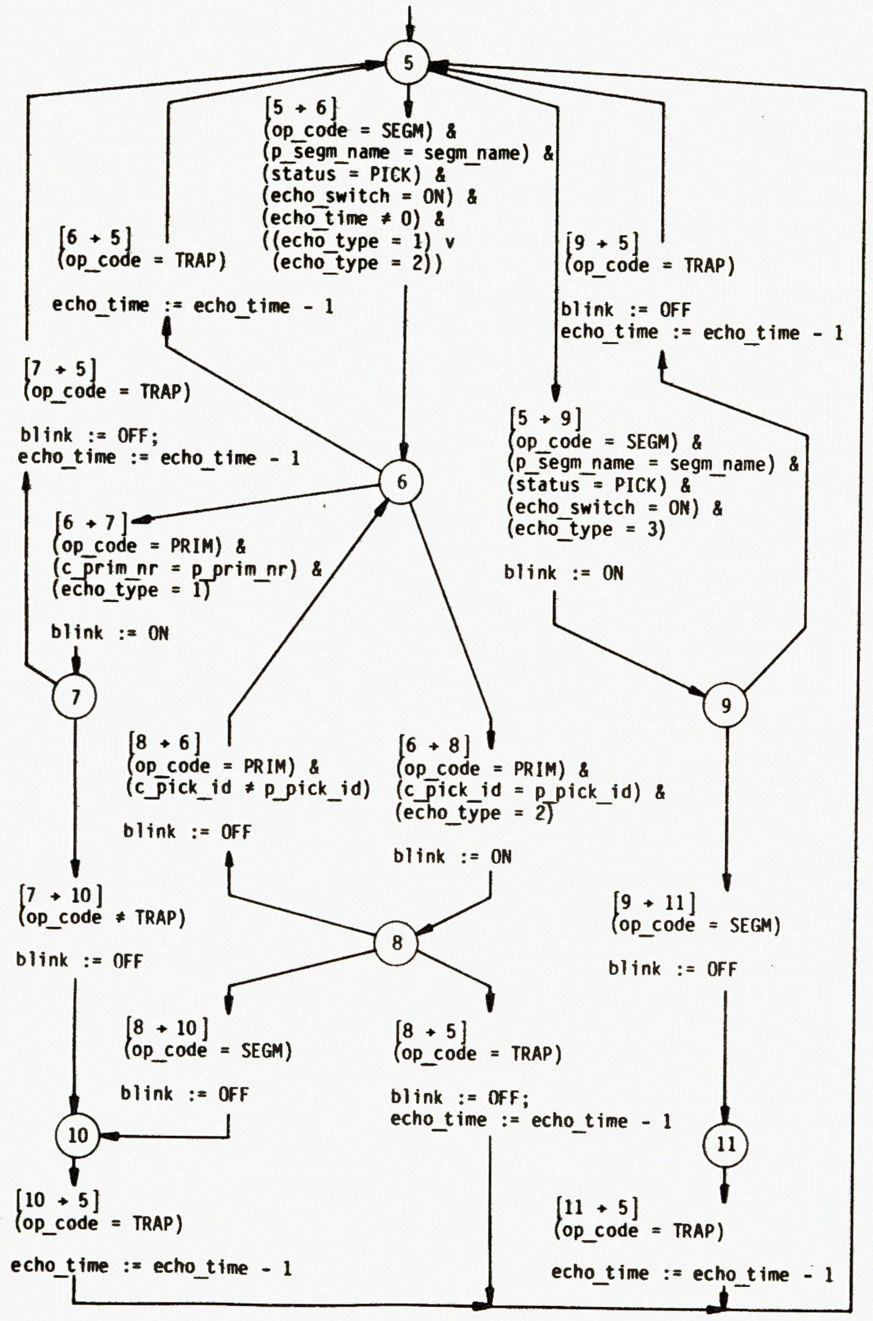
5
[5 → 6] (op_code = SEGM) & (p_segm_name = segm_name) & (status = PICK) & (echo_switch = ON) & (echo_time ≠ 0) & ((echo_type = 1) v (echo_type = 2))
[6 → 5] (op_code = TRAP)
echo_time := echo_time - 1
[9 → 5] (op_code = TRAP)
blink := OFF
echo_time := echo_time - 1
[7 → 5] (op_code = TRAP)
blink := OFF;
echo_time := echo_time - 1
6
[5 → 9] (op_code = SEGM) & (p_segm_name = segm_name) & (status = PICK) & (echo_switch = ON) & (echo_type = 3)
blink := ON
[6 → 7] (op_code = PRIM) & (c_prim_nr = p_prim_nr) & (echo_type = 1)
blink := ON
7
9
[8 → 6] (op_code = PRIM) & (c_pick_id ≠ p_pick_id)
blink := OFF
[6 → 8] (op_code = PRIM) & (c_pick_id = p_pick_id) & (echo_type = 2)
blink := ON
[7 → 10] (op_code ≠ TRAP)
blink := OFF
[9 → 11] (op_code = SEGM)
blink := OFF
8
[8 → 10] (op_code = SEGM)
blink := OFF
[8 → 5] (op_code = TRAP)
blink := OFF;
echo_time := echo_time - 1
10
11
[10 → 5] (op_code = TRAP)
echo_time := echo_time - 1
[11 → 5] (op_code = TRAP)
echo_time := echo_time - 1

Figure 4b. Pick Algorithm - Echo Performance.

primitive within the segment has to be initialized (c_prim_nr := 0).
- When a pick identifier is encountered (op_code = PICK_ID) the corresponding pick identifier has to be remembered (c_pick_id := pick_id).
- When an output primitive is encountered (op_code = PRIM) the current number of the primitive within the segment is augmented (c_prim_nr := c_prim_nr + 1).

Before interaction may start, initial values which have been provided by an application have to be checked for validity. Here we will only outline how this is achieved. If the initial status is PICK, the pick algorithm has to proceed the entire display list, whereby it has to check (a) if a segment according to the initial segment name is included in the display list and is detectable, and (b) if this segment contains at least one primitive corresponding to the initial pick identifier. When the check fails, the initial status is set to NOPICK.

State_1 is the state of the pick automaton after interaction has started or after a measure has been obtained. Whenever state_1 is entered, the light-pen interrupt flip-flop is cleared (lp_int := FALSE) and echo is turned off (blink := FALSE). State_1 is left when a light-pen interrupt is encountered at the end of a refresh cycle (lp_int = TRUE) & (op_code = TRAP).

In state_2 the next visible and detectable segment (which due to reasons explained in the previous section is always the segment with the relative highest priority which has not been investigated yet) is searched. When a visible and detectable segment is found (op_code = SEGM) & (visib = TRUE) & (detec = TRUE), the light-pen interrupt flip-flop is cleared and state_2 is left.

In state_3 has to be investigated whether an interrupt from the light-pen has occurred during the output of the last primitive. When this is the case (lp_int = TRUE) the name of the segment the primitive belongs to, the current pick identifier and the current number of the primitive in the segment are remembered. These operations are performed before evaluation of the parameters of a new function will alter the current values. The status of the measure is set to PICK. When no interrupt has been registrated the next segment is inspected. When this segment is invisible or undetectable the automaton will return to state_2. When in state_2 or state_3 the end of the refresh cycle is reached the automaton has failed to find a picked primitive, the status of the measure is set to NOPICK. This may occur when the user has pointed at a non-retained primitive, a primitive in an undetectable segment, or at a device echo.

In state_4 the end of the current refresh cycle is expected without further activities.

Before entering state_5 echo time has been initialized to an implementation dependent value (echo_time := TIME). When echoing is on (echo_switch = ON) the echo process is started. When the status of the measure is NOPICK a dummy echo is performed until echo time has elapsed (echo_time = 0). When the segment containing the picked primitive is encountered (op_code = SEGM) & (p_segm_name = segm_name), echoing is performed in dependence from the echo type. For echo type 3 blinking is immediately turned on and remains on until output of the segment has terminated.

In state_6 the picked primitive (for echo type 1) or all primitives of the segment with the same pick identifier (for echo type 2) are searched.
- For echo type 1 the current number of the primitive within the segment has to be compared to the number of the picked primitive (op_code = PRIM) & (c_prim_nr = p_prim_nr). When the test succeeds blinking is turned on.
- For echo type 2 the current pick identifier is compared to the pick identifier of the picked primitive (op_code = PRIM) & (c_pick_id = p_pick_id). When they are equal blinking is turned on.
When the trap function is encountered in state_6, the automaton immediately returns to state_5.

In state_7 the echo function for echo type 1 is terminated, blinking is turned off. When the end of the refresh cycle is reached, the automaton returns to state_5.

In state_8 the echo function for echo type 2 is either suspended when the pick identifier changes (in this case state_6 is reentered) or terminated when a new segment or the end of the refresh cycle are encountered (in this case the automaton goes back to state_5).

In state_9 the echo function for echo type 3 is terminated. When the end of the refresh cycle is reached, the automaton returns to state_5.

In state_10 and state_11 the end of the current refresh cycle is awaited. After that the automaton returns to state_5.

The algorithm is essentially self-contained but for the propagation of the measure. The measure is complete when state_5 is reached. The identification of a graphical item requires no more than two refresh cycles: one cycle in which the first impulse from the light-pen is recognized, and one cycle in which the display list is scanned to find the picked primitive. When state_5 is entered the UP is interrupted and may read the measure from the dual-port memory. Any further interpretation of the measure (including mapping to choice integers) is performed by the UP. The actual implementation of the algorithm has been accomplished with the help of a universal meta-microassembler and some hardwired logic for testing purposes.

7 Alternatives

Two aspects of device independent graphics systems have become a bone of contention:
- The logical input device concept does not permit an application to address or change device properties on a lower level. Thus interaction techniques are ruled out, which depend on the presence and/or behaviour of a particular device.
- An application is inhibited to bypass logical input devices, as device independent graphics systems assume full power over all graphical resources.

Problems encountered when an interaction technique depends on the presence of a specific device are discussed in [2]. The logical pick device concept has attracted some criticism due to the fact, that in some applications a two-level identification of graphical entities is not sufficient. Typed picking (also considered a remedy for the resolution of picking ambiguities) has been proposed instead [10], [11]. The designers of user interface management systems and "screen handlers" have to cope with another problem: Who assumes the responsibility for graphical interactions, the application, the graphics system, or the interface manager ? Subjects concerned in this context are event queue management, provision of acknowledgement, graphical rubout, et al.

Our display processor is suited for the implementation of a user interface management system with internal control [14]. Effects on the pick algorithm should be within reasonable boundaries. Typed picking would occupy some additional registers in the dual-port memory for storing types of picked entities, the pick algorithm should become only slightly more complicated. The introduction of picking hierarchies would require to alter the present structure of the display list with minor consequences for the pick algorithm. Keeping away particular interaction techniques or styles from the microprogram level, should assist in the adaptation of the pick algorithm to future expansions.

References:

(Reference [1] is not cited in the text)

[1] Graphical Input Interaction Technique (GIIT), Workshop Summary. Computer Graphics 17, 1 (Jan. 1983).

[2] Buxton, W. Lexical and Pragmatic Considerations of Input Strucutures. In: [1], pp. 31-37.

[3] Status Report of the Graphics Standards Planning Committee. Computer Graphics 13, 3 (Aug. 1979).

[4] Dwyer, B. A User-Friendly Algorithm. Comm.ACM 24, 9 (Sept. 1981), 556-561.

[5] Draft International Standard ISO/DIS 7942, Information Processing Graphical Kernel System (GKS), Functional Description, Version 7.2, NI-5.9/1-83, Nov. 1982.

[6] Green, M. A Catalogue of Graphical Interaction Techniques. In: [1], pp. 46-52.

[7] Jacob, R.J.K. Using Formal Specifications in the Design of a Human-Computer Interface. Comm.ACM 26, 4 (April 1983), 259-264.

[8] Kasik, D. Software Tools and Techniques. In: [1], pp. 20-24.

[9] Leich, H., Levchanovsky, F.V., and Prikhodko, V.I. A Multi-Microprocessor Based Intelligent Graphics Terminal. Microprocessors and Microprogramming 12 (1983), 175-180.

[10] Olsen, D.R., and Dempsey, E.P. SYNGRAPH: A Graphical User Interface Generator. Proc. of SIGGRAPH '83, Computer Graphics 17, 3 (July 1983), 43-50.

[11] Olsen, D.R. Automatic Generation of Interactive Systems. In: [1], pp. 53-57.

[12] Rosenthal, D.S.H., Michener, J.C., Pfaff, G., Kessener, R., and Sabin, M. The Detailed Semantics of Graphics Input Devices. Proc. of SIGGRAPH '82, Computer Graphics 16, 3 (July 1982), 33-38.

[13] Rosenthal, D.S.H. The GKS Input Facilities and How To Use Them. Computer Graphics Forum 2, 2/3 (Aug. 1983), 97-103.

[14] Rosenthal, D., and Yen, A. User Interface Models Summary. In: [1], pp. 16-20.

[15] Rudalics, M. An Intelligent Graphics Terminal's Intermediate Data Base. Proc. of Eurographics '83, North-Holland Pub., (1983), 383-392.

Received by Publishing Department
on December 12, 1984.

Науманн Б., Рудалич М. E11-84-770

Программное обеспечение светового карандаша

Недавние усилия в стандартизации графического программного обеспечения привлекли все усиливающееся внимание со стороны разработчиков графических приборов. Эта тенденция лучше всего иллюстрируется разработкой рабочих станций с встроенными CORE и GKS функциональными возможностями. Стандарт GKS также оказал влияние на разработку интеллектуального графического терминала /ИГТ/ в ОИЯИ. Определение и реализация pick device как средства ввода, которое используется для выбора графических объектов на экране дисплея конкретно для ИГТ, является предметом данной статьи. Определение базируется на оценке соотношения концепций "абстрактный" вывод и "логический" ввод. Идентификация графического объекта и обеспечение обратной связи к пользователю представляется в форме диаграммы состояний перехода. Подход, заключающийся в применении техники формального определения, оказался весьма подходящим при реализации алгоритмов в микрокоде.

Работа выполнена в Лаборатории вычислительной техники и автоматизации ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна 1984

Naumann B., Rudalics M. E11-84-770

A Microprogrammed Pick Device

Recent efforts in the standardization of graphics software have received increasing attention from the designers of graphics devices. This trend is best exemplified by the development of workstations with inbuilt CORE and GKS functionalities. GKS has also influenced the design of an intelligent graphics terminal (IGT) at the Joint Institute for Nuclear Research. Specification and implementation of a pick device - an input tool which is used to select graphical entities on a display surface - for the IGT are subject of this paper. Specification has been based on an evaluation of the interrelated concepts of "abstract" output and "logical" input. The indentification of a graphical item and the provision of feedback to the user are presented in form of a state transition diagram. The approach to use a formal specification technique has proven to be of great assistance in the actual implementation of algorithms in microcode.

The investigation has been performed at the Laboratory of Computing Techniques and Automation, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna 1984