stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 202/82 AUGUSTUS

D.S.H. ROSENTHAL, J.C. MICHENER, G. PFAFF
R. KESSENER & M. SABIN

THE DETAILED SEMANTICS OF GRAPHICS INPUT DEVICES

Preprint

kruislaan 413 1098 SJ amsterdam

BIBLIOTHEEK MATHEMATISCH CENTRUM
AMSTERDAM

Printed at the Mathematical Centre, 413 Kruislaan, 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.).

CR Categories and Subject Descriptors: I.3.4 [Computer Graphics]: Graphics Utilities — graphics packages; I.3.6 [Computer Graphics]: Methodology and Techniques — device independence; interaction techniques

General Terms: Standardization

The Detailed Semantics of Graphics Input Devices

by

David S. H. Rosenthal
James C. Michener[1]
Günther Pfaff[2]
Rens Kessener[3]
Malcolm Sabin[4]

ABSTRACT

The concept of *virtual input devices,* enunciated by Wallace, has been the accepted basis for producing device-independent interactive graphics systems. It was used by GSPC for the Core System, and it underlies the draft international standard GKS.

During the recently concluded technical review of GKS, the input facilities became a bone of contention. The discussions revealed many inadequacies in the virtual input device concept, and were finally resolved using a refined and extended model of input, which is presented here by some of the participants in the discussions. Examples are included, showing how the GKS facilities derive from the model, and the Core's "STROKE" device is used to show how the model controls future extensions to GKS. The model is also used to describe the other differences between the input facilities of the Core System and GKS.

This paper was presented at the ACM SIGGRAPH '82 conference in Boston, Mass., on July 28th 1982. It appears in the Proceedings as published in *Computer Graphics*. The other authors' affiliations are:

1. Intermetrics Inc., Cambridge, Mass., U. S. A.
2. Technische Hochschule Darmstadt, Federal Republic of Germany.
3. Technische Hogeschool Eindhoven.
4. CAD Centre, Cambridge, England.

1. Introduction

> "There is a theory which states that if ever anyone finds out exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable."
> *"The Restaurant at the End of the Universe"* by Douglas Adams

On a considerable basis of earlier work[5, 2, 8], Wallace enunciated the concept of virtual input devices[10] as a means whereby interactive graphics applications could be insulated from the peculiarities of the input devices of particular terminals, and thereby become portable. The idea was that the applications programmer had available a range of virtual input devices, the only visible aspect of which was the type of the value they returned. Thus, if a position were required, a LOCATOR device would be selected, if a character string were required, a STRING device would be selected, and so on.

From the start, it was evident that the pure virtual device concept was inadequate. Virtual devices needed other visible aspects, controlling details of the operator interface such as echoing. This *logical* device concept formed the basis for the GSPC Core System[9], and it was also used by the initial versions of the draft international standard GKS[3]. During the recently concluded technical review of GKS by a working group of the International Standards Organisation, the input facilities became a bone of contention. Criticisms of these versions of GKS concentrated on:

- The precise data types to be returned by the different classes.
- The different levels of detail at which different kinds of input behaviour were specified.
- The lack of uniformity among the different logical device classes as to the details of their behaviour.
- The lack of clear distinction between the concepts of:
  - *Simulating* a logical device using particular types of hardware.
  - *Prompting* an operator for input.
  - *Echoing* an operator's actions.
  - *Acknowledging* an operator's generation of events.
- The difficulty of relating any of these "output" concepts to logical input devices.

It became clear that these problems were all related to deficiences in the underlying model of input, and the discussions were finally resolved by a refined and generalised concept of logical input devices, which is described below.

2. The Model

An application program obtains input from a number of *logical input devices*, divided into *classes* according to the type of data obtained. Typical classes are LOCATOR, VALUATOR, PICK, CHOICE, and STRING.

2.1. Application Program Use of Logical Input Devices

At the highest level, a logical input device is either *accessible* to the application program, or it is *inaccessible*. Initially, all devices are inaccessible. If a device is to be used for input, it must first be *acquired*, that is, a connection must be made between the identifier known to the application program and the external name(s) for the physical device(s). Once this connection is made, the device becomes accessible and interactions may take place using the device. The device may also be *released*, breaking the connection between the identifier and the physical devices. The device is then inaccessible until it is reacquired.

There are three generally accepted ways in which application programs can obtain data from logical input devices; they are known as different *modes* of operation:

- In SAMPLE mode the application program invokes a function to obtain the current logical data value from the device.

— In EVENT mode the operator's actions create *events*, records of the logical data value of the device at a specific moment in time; the system preserves these records in one or more *event queues*, the contents of which the application program can process at its convenience.

— In REQUEST mode the application program invokes a function permitting the operator to adjust the logical data value of the device and then indicate that the value is satisfactory; the function waits until this has been done and then returns the value.

Initially, when acquired, a device is in REQUEST mode. When a device is in REQUEST mode, but is not currently the subject of a REQUEST function invocation, the device is not available to the operator. This corresponds to what has been called a *disabled* state, although the term seems to be dropping from use.

A logical input device is taking part in an *interaction* while the device is in SAMPLE or EVENT mode, or while the device is the subject of a REQUEST function invocation. The term *enabled* has been used for this device state.

Note that the definitions of these modes refer to two concepts, the current logical data value, and specific moments in time at which the operator indicates that the current value is important. These concepts are described more fully in the next section.

2.2. Measures and Triggers

All logical input devices, being abstractions, are considered to be "simulated" by an implementation, even if the mapping from physical device(s) to a logical device is quite direct. The simulation of a logical input device has two major parts, the *measure* part and the *trigger* part. The measure part determines how the operator controls the logical data value, and the trigger part determines how the operator indicates that the current value is important.

The measure part of a logical input device is only active when the device is taking part in an interaction. At these times, an independent *measure process* is (conceptually) in existence that uses the states and changes of state of various physical input devices to control a logical data value (the device's *measure*) appropriate to the logical device's class. It is permitted that a single physical input device simultaneously affect the measures of several logical input devices; nevertheless, the measure processes are still considered distinct.

The trigger part of a logical input device responds to changes in the state of physical input devices either by remaining quiescent,* or by *firing*. When the trigger of a logical input device fires, it sends a message to the measure process for the device. The effect of the signal depends upon the mode of the device, and is explained below.

Several logical input devices may have their trigger part in common. Whenever any of the devices sharing a particular trigger part are taking part in an interaction, a single *trigger process* for that trigger is (conceptually) in existence. The firing of the trigger signals all the appropriate measure processes. It is a requirement that a single operator action cause the firing of no more than one trigger. Thus, in contrast to measure processes (which must be unique to a logical device), it is normal for trigger processes to be shared between logical devices.

When a measure process receives a signal from the trigger process of its logical device, the actions it takes depend upon the device's mode as follows:

— In REQUEST mode, the measure process returns its data value to the application, and then dies.

— In SAMPLE mode, signals from triggers are always ignored. The application may obtain the data value from the measure process whenever it pleases.

— In EVENT mode, the measure process attempts to add an *event record*, containing its identification and data value, to the appropriate input queue. Other

* Its invisible internal state may change.

measure processes using the same trigger process may also be attempting to add records to the same queue at the same time. The resulting *group of simultaneous event records* must either all be added to the queue, or none must be added to the queue. If the measure processes fail to add their events to the queue, input queue overflow must be reported for that queue.

Groups of simultaneous event records are marked as such; when an event record is dequeued, the application must be able to discover whether more events in the same group remain in the queue.

2.3. Attributes of Logical Input Devices

When accessible to the application program, a logical input device has certain characteristics or *attributes* that distinguish it, in a general fashion, from other logical input devices of the same input class. Depending on the particular graphics system, some attributes will be under application control, while others will have been fixed by the implementor. Some of the attributes of logical input devices are:

— Current mode of operation.

— How the implementation simulates the logical device using physical devices.

— How the operator is informed that a measure process has come into existence, and thus that its associated physical devices are available for manipulation. This is called the *prompt*.

— How the operator is informed of the logical device's measure (its logical data value). This is called the *echo*.

— How the operator is informed of a significant firing of the input device's trigger; this is called *acknowledgement*. A significant trigger firing is one satisfying a REQUEST function invocation, or adding events to the queue,

— An initial value, of the type appropriate to the class, for use by the device's measure process when it comes into existence.

— A switch turning echo on or off.

The attributes may also contain extra information used, for example, by particular simulation, prompting and echoing techniques. Note, however, that the device's measure is not considered an attribute.

2.4. The Life Cycle of an Input Device

It is now possible to outline the sequence of operations that corresponds to a specific logical input device taking part in a interaction:

— A measure process is created for the device, and its value is set to the initial value in the device's state.

— If the trigger process for the device is not in existence, it is created.

— The operator is prompted for input, using the selected technique.

— If the echo switch in the device's state is on, echoing is commenced, using the selected technique.

— As the operator manipulates the physical input devices, the trigger may fire, causing the appropriate one of the set of actions outlined above, depending on the device's mode.

— If a trigger firing is significant, it is acknowledged.

— Eventually, either because in REQUEST mode the trigger fires, or because the device leaves SAMPLE or EVENT mode, the measure process dies.

— When a trigger process has no measure processes, it also dies.

3. Applying the Model to GKS

As an example of the use of the model, we take the input facilities of GKS[7]. A fundamental concept of GKS is the *workstation* [4], a collection of input and output facilities, treated as a unit by the application program, forming a single logical channel of communication to the user. An application may drive many workstations, several of which may support one or more logical input devices. However, there is a *single* event queue shared by

all workstations.

Each logical input device is treated as part of a particular workstation, and is acquired and released as its workstation is opened and closed. The attributes of each logical device are part of the Workstation State List for the corresponding workstation. The application program name for a logical input device (shown below as ID) is a pair, thus:

<Workstation identifier, Device identifier>

The implementor of the workstation selects for each logical device a single technique by which it is simulated using the available hardware. The implementor may provide different simulations as different logical devices in the same class, but GKS does not permit the application program to change individual simulation techniques.

3.1. GKS Modes

All GKS logical devices can operate in each of the three modes, REQUEST, SAMPLE, and EVENT. By default, devices are in REQUEST mode. Given this and omitting some details, the set of input functions becomes at least:

— Operations on the device's attributes:

INITIALISE <class>(ID, INITIAL_VALUE)
SET <class> MODE(ID, MODE)

— Input directly from the device:

REQUEST <class>(ID, VALUE)
SAMPLE <class>(ID, VALUE)

— Input from event queue:

AWAIT EVENT(ID, CLASS)

— Examination of most recently awaited event record:

GET <class>(VALUE)

— Detection of simultaneous events:

INQUIRE MORE SIMULTANEOUS EVENTS(FLAG)

An interaction with a device starts whenever REQUEST <class>, or SET <class> MODE with MODE=EVENT or MODE=SAMPLE, is invoked. At this point, the measure process is (re-)created and initialised to the value from the workstation state list.

3.2. GKS Device Classes

GKS provides five device classes, LOCATOR, VALUATOR, CHOICE, PICK, and STRING. The Core System, following Wallace, considers BUTTON more appropriate as a *primitive* input class, than CHOICE. The Core System provides an additional class, STROKE, which is used below as an example of how the model controls possible extensions to GKS.

3.2.1. LOCATOR

Both GKS and the Core use a two-stage process to transform from the *world* coordinates used by the application to the individual *device* coordinates used by each display. Coordinates are first transformed to a single space shared by all devices, called *normalised device coordinates* (NDC) by the window/viewport transformation. Then each workstation has a private transformation from NDC to its own device coordinates.

Wallace originally suggested that LOCATOR devices returned a position in device coordinates. The Core System's LOCATOR devices return a position in NDC. The application eventually needs a position in world coordinates, since these are used for output. The difficulty in providing world coordinates lies in selecting the window/viewport transformation whose inverse is to be applied. Consider an application in which a view of a drawing, a part of a symbol library, and some menus share the screen. Each was created using a different window/viewport transformation; the operator's actions may require the application to change any of these views, and thus to re-establish an appropriate transformation for drawing them.

The transformation to be used cannot simply be that active for output, which must be set according to output requirements, and changed even while devices are in EVENT mode. The Core System's NDC locators avoid this problem, leaving it to the applica-

tion to select an appropriate transformation.

GKS takes an alternative approach, providing multiple window/viewport transformations,* referred to by an index. For output, the application selects one using its index. For input, the application arranges the transformations in a priority order. When a physical locator returns a coordinate, the workstation transforms it back to NDC, and then transforms to world coordinates by:

— Scanning the list of transformations in decreasing priority order, until the NDC position lies inside the viewport of a transformation.

— Using the inverse of this transformation to provide a world coordinate value.

— Returning as the measure both the world coordinate value and the index of the selected transformation.

In this way, the locator itself selects an appropriate transformation. In the example above, if the locator's device coordinate position lies within the part of the screen showing the drawing, the drawing's window/viewport transformation will be used. If it is in the part showing the symbol library, the library's transformation will be used. The application knows which was used, because the index of the transformation used is part of the measure. In general, there will be enough transformations to assign one to each part of the screen in use, so that they will only need to be changed infrequently. In any case, there is a default transformation that cannot be changed, in effect returning NDC if no other transformation can be found.

3.2.2. VALUATOR

GKS provides a classical VALUATOR class, whose measures are real values in ranges specified on a per-device basis by the application.

* Termed *normalisation* transformations, since they transform to normalised device coordinates.

3.2.3. CHOICE

GKS provides a CHOICE class, whose measures are either integers up to a device-specific limit, or an indication of "no choice". No choice might, for example, be a state in which no buttons on a button box were depressed. The class is intended to provide a "menu" capability, and has potentially complex application-controlled prompting techniques, including the display of a menu consisting of strings or the primitives in a segment.

3.2.4. PICK

GKS provides a PICK class, whose measures are either segment name and pick identifier pairs, or an indication of "no pick". No pick might, for example, be a state in which the light pen was not pointing at any detectable segment.

3.2.5. STRING

The measures of GKS STRING class devices are (possibly null) strings of characters. The operator is presented with the initial string, and a cursor at an application-specified position within it. Replacement of characters starts at the cursor position, and may extend the string up to an application-specified maximum length.

3.3. A Possible STROKE Device Class

GKS does not provide a STROKE device class. However, using the model it would be easy to design one. The design would proceed in three stages. First, omitting some details, the functions required are:

INITIALISE STROKE(ID, INITIAL_VALUE)
SET STROKE MODE(ID, MODE)
REQUEST STROKE(ID, VALUE)
SAMPLE STROKE(ID, VALUE)
GET STROKE(ID, VALUE)

Secondly, the data type appropriate to the class is determined. The measure of a STROKE device is a (possibly null) string of positions in world coordinates, and a normalisation transformation number. The coordinates of the returned polyline are re-transformed by the inverse of the window/viewport transfor-

mation of highest priority in whose viewport they all lie; the index of this transformation is part of the value.

Because their measures are both values resulting from a *sequence* of operator actions, the STROKE class behaves analogously to the STRING class, in that it takes an initial stroke and a cursor position within it. Replacement of strokes starts at the cursor position and may extend the polyline up to an application-specified maximum. Details such as whether the individual positions of the stroke are triggered by distance, time, or operator action, and how the operator "rubs-out" erroneous positions, are left to the workstation implementor.

3.4. Prompting and Echoing

The details omitted from the descriptions above concern prompting and echoing. For each class, GKS defines several prompt/echo techniques. At least one very simple technique must be supported for every device. When a device is initialised, a particular prompt/echo technique is requested, and appropriate parametric information is supplied. These attributes include an echo area, which the technique may use to display the prompt or echo, and a data record containing device- and implementation-specific information such as an array of strings for a CHOICE device using text menus.

3.5. Differences Between GKS and the Core

Although they are conceptually similar, there are some detail differences between the input facilities of GKS and the Core. The model is equally useful for describing the Core, though in this section we only describe the differences.

Because the Core has no workstation concept, it has explicit functions for acquiring and releasing logical input devices, for example INITIALIZE_DEVICE. Note that this does *not* provide an initial value for the device; the Core has individual functions for setting particular attributes of logical input classes, and the only classes for which an initial value setting function is provided are LOCATOR and VALUATOR.

The Core recognises only SAMPLE and EVENT modes. Each device class operates only in one mode; LOCATOR and VALUATOR in SAMPLE mode, and PICK, KEYBOARD, BUTTON, and STROKE in EVENT mode. The measure and trigger processes are created by an ENABLE_DEVICE invocation, and destroyed by a DISABLE_DEVICE invocation.

Facilities are provided to *associate* one or more LOCATOR or VALUATOR devices with a device in an "event" class. An association between a device in a "sample" class and a device in an "event" class in effect creates a new logical device, operating in event mode, which has the measure of the "sample" device* and the trigger of the "event" device.

An group of associated devices share the same trigger (that of the "event" device) and so generate a group of simultaneous events. Unlike GKS, the Core combines all the reports in a group of simultaneous events into a single complex report; the firing of a single trigger can place at most one report in the queue.

In GKS, the creation of groups of associated devices is the preserve of the workstation implementor. Facilities to provide application control over associations would require the addition of two new functions (and the corresponding inquiries) to GKS:

ASSOCIATE(ID1, ID2)
DISSOCIATE(ID2)

The effect would be to disconnect the measure of device ID2 from its trigger, and to connect it to the trigger of device ID1 until it was dissociated. When the trigger of device ID1 fired, the resulting group of simultaneous events would contain an event from device ID2.

* It is thus in the class of the "sample" device.

4. Implications of the Model

Because there was otherwise no exit from a REQUEST except by supplying a valid value, GKS provides a "break" facility. This permits an operator, when REQUESTed for a value, to refuse to supply one. It provides, among other capabilities, an easy way for the operator to indicate "end-of-input".

Another implication of the model may reflect on the current discussions of graphics virtual device interfaces. The model now insists that the essential preliminary to any input operation is an output operation providing an initial value of the appropriate type. This strongly encourages a symmetric approach to incorporating input into a virtual device definition, insisting that the responses from input devices are similar to output commands.

This symmetry is enhanced by the observation that the prompt/echo information now behaves in effect as the attributes of an input primitive, modifying its visible appearance in a workstation-dependent fashion.

5. Conclusion

Since it was proposed, the concept of virtual input devices has been extremely useful, but has also attracted severe criticism[6, 1] from, among others, one of us. Although this refined model answers some of these attacks, the fundamental problems brought to light by the critics are still present. Nevertheless, it is clear that the time is not yet ripe for a standard, whose rôle is to codify existing good practice, to incorporate a more radical approach to input.

We expect that the generalised and rejuvenated concept of logical input devices will remain the basis for the device-independence of interactive graphics applications for some considerable time. However, now that they have a more robust and detailed target, we would welcome renewed attention from the critics.

6. Acknowledgements

Our grateful thanks are tendered to all those who took part in the discussions on input in the Draft Standards Subgroup of ISO TC97/SC5/WG2, and in the national discussions supporting them. Particular thanks are due to Paul ten Hagen of the Stichting Mathematisch Centrum, Amsterdam, to Ray Spiers, to Dick Puk, and to Marceli Wein of the National Research Council of Canada, who was still willing to talk to us, even after being thrown in at the deep end.

David Rosenthal was supported in this work by Science and Engineering Research Council grants N2B 1R 0371 and GR/A80341.

REFERENCES

[1] E. Anson, "The Semantics of Graphical Input," *Computer Graphics* **13**(2), pp.113-120 (August 1979).

[2] I. W. Cotton, *Network Graphic Attention Handling,* Proc. Online '72 Conf., Brunel University, Uxbridge, England (September 1972).

[3] DIN, "Graphical Kernel System (GKS) — Functional Description," (Version 6.6) (May 1981).

[4] J. Encarnação, G. Enderle, and others, "The Workstation Concept of GKS and the Resulting Conceptual Differences to the GSPC Core System," *Computer Graphics* **14**(3), pp.226-230 (July 1980).

[5] J. D. Foley and V. L. Wallace, "The Art of Natural Man-Machine Communication," *Proc. IEEE* **62**(4), pp.462-471 (April 1974).

[6] R. A. Guedj and others (eds.), *IFIP Workshop on Methodology of Interaction,* (publishers North-Holland), Seillac, France (May 1979).

[7] ISO, "Graphical Kernel System (GKS) — Functional Description," ISO DP 7942 (January 1982).

[8] W. M. Newman, "A System for Interactive Graphical Programming," *AFIPS Conference Proceedings (SJCC)* **32**, pp.47-54 (1968).

[9] SIGGRAPH-ACM (GSPC), "Status Report of the Graphics Standards Planning Committee," *Computer Graphics* **13**(3) (August 1979).

[10] V. L. Wallace, "The Semantics of Graphics Input Devices," *Computer Graphics* **10**(1) (Spring 1976).

MC NR

35224

ONTVANGEN 8 SEP. 1982