Pointing at items on a graphics display is one of the most useful methods of interacting with a system graphically. This paper examines existing graphical support and lists requirements for high-level support of graphical interaction. The architecture of a prototype system with high-level support for graphical interaction is presented. This includes database support for manipulating graphical data and device-independent graphical support based on a proposed standard for graphical interaction. Algorithms are presented for identifying items selected from a display by the user. Inclusion of a database management system in graphical software support is shown to be helpful in meeting the requirements of interactive graphical application programs.

Software architecture for graphical interaction

by D. L. Weller, E. D. Carlson, G. M. Giddings, F. P. Palermo, R. Williams, and S. N. Zilles

Interactive graphics is made possible by hardware and software support for graphical input. The ACM Graphics Standard Planning Committee (GSPC) has made its Core proposal for a graphics standard[1] that identifies the following six types of logical input devices:

- *Keyboards* for typing alphanumeric data.
- *Buttons* for program function activation.
- *Stroke devices* for direct visual graphics entry (e.g., electronic tablet).
- *Valuators* for analog quantity entry (e.g., dials and meters).
- *Locators* for position entry (e.g., joysticks).
- *Picks* for item selection (e.g., light pens and joysticks).

Of these six device types, picks and locators may be the most useful for interactive graphics because they allow one to interact directly with a graphical output by pointing. Foley and Wallace[2] have observed that graphical interaction by pointing is easy and natural for most people. Our experience with picks and locators

in a variety of applications[3–7] indicates that their use improves the human factors of graphics applications, especially applications designed for nonprogrammers. In this paper we discuss picks, but many of the concepts and algorithms apply equally to locators.

Picks are also important because they can simulate other devices. They can be used to select text strings or symbols from a menu to simulate the function of buttons and keyboards. Picks can be used to position a tracking cross, thereby simulating the function of a locator. They can also be used to move an indicator on a graphically displayed dial, thus simulating the function of a valuator. A powerful use of picks is for graphical query. For example, a pick can be used to select a subset of a picture to be redisplayed with more detail at a larger scale.

Returned with each pick is a *tag*, which is a unique identifier of a graphical item. Tags are also known as pick identifiers, pick attributes, or correlation values. Support for picks requires two basic functions: identification and linking. *Identification* determines the tag of the picked graphical item. The *linking* function causes an action based on the returned tag. A tag may link a variable, data, or the name of a module in an application program to a graphical item and thus provide meaning for the item picked.

In the next section we summarize the support for pick devices in the GSPC Core proposal because that proposal is representative of existing graphics systems. Following that we describe application requirements for using picks and show that there is a need for higher-level support than that provided by existing graphics systems. Presented next is the architecture of a prototype system, the Picture Building System (PBS),[8–11] for supporting graphical interaction. The prototype is based on a database management system. Following that section we describe pick identification algorithms. In the final section we show the simplification of pick support by the use of a database management system.

Pick support in existing graphics systems

the pick function

The pick devices provided by existing graphics hardware are primarily light pens and cursors controlled by joysticks, tablets, and track-balls.[12,13] A pick device is typically used by pointing to an item on a screen and pressing a button or switch. When this happens, a pick identification function determines the item picked, and the tag of the picked item is returned to the application program.

The returned tags differ from system to system, depending on the type of terminal. For terminals with a refresh buffer, the buffer address is often used as a tag. In this case, the identification func-

tion is performed by the hardware,[14] and the linking function by the software. For other terminals, the tag is often the *x-y* coordinates of the pick, in which case the software performs both the identification and the linking functions.

Some graphics subroutine packages provide additional pick support. The *x-y* coordinates and programmer-assigned tags of *detectable* items (items that can be picked) are stored in a *pick table*. When a pick occurs, an identification algorithm uses the pick table along with a *pick window* (which is an area based on the specified *x-y* position) to determine the item picked and returns the corresponding tags.

Some systems provide tags for all items in the pick window, whereas others provide only a single tag for the first item. In some systems,[15] tags are stored in the refresh buffer, in a separate buffer, or in a structured display file. If a structured display file is used, a vector of identifiers corresponding to the hierarchical structure of the picked item may be used as a tag. If a separate pick table is not kept, the data for the entire picture must be regenerated whenever a pick occurs. This is generally much slower than searching a pick table, which has entries for the detectable items only. In summary, tags can be buffer addresses, *x-y* coordinates, programmer-assigned identifiers, or vectors of identifiers.

interface for pick support

A summary of the support for picks provided by ten graphics software packages is given in a survey performed by the GSPC.[12] The GSPC Core proposal[1] specifies commands for pick support. Although the proposal has not yet been adopted as a standard, we use it here as an example of software support for picks.

The PICK ID, a programmer-assigned integer, is statically bound to a primitive, such as a text string, line, or point (marker), when that primitive is added to a segment. Primitives can be grouped into named segments, which can have attributes that apply to all the primitives in the segment. *Detectability* and *visibility* are dynamic attributes that determine whether an item may be picked. If a segment is detectable, the pick device can be used to select primitives (that have a PICK ID) within that segment. If the segment is nondetectable, a pick cannot detect any primitives in the segment (even if the primitives have a PICK ID). Segments can be given a detectability priority that can be used to resolve pick ambiguities (as discussed later in this paper). Visibility affects detectability because primitives in a segment that is invisible cannot be detected by a pick. Commands, such as those shown in Table 1, are provided to the programmer to manipulate PICK IDs and pickability.

The proposed GSPC Core commands for controlling pick activation include initialization, termination, the enabling and disabling

Table 1 Commands used to manipulate PICK ID and pickability

Command	*Function*
SET PICK ID (id)	set PICK ID to id
SET SEGMENT DETECTABILITY (segment name, id)	set detectability to id
SET SEGMENT VISIBILITY (segment name, on/off)	set visibility on or off
INQUIRE PICK ID (id)	determine current PICK ID
INQUIRE SEGMENT DETECTABILITY (segment name, detectability)	determine detectability
INQUIRE SEGMENT VISIBILITY (segment name, visibility)	determine visibility

Table 2 Commands for a single pick device

Command	*Function*
INITIALIZE DEVICE (pick, number)	initialize a specific pick device
TERMINATE DEVICE (pick, number)	terminate a specific pick device
ENABLE DEVICE (pick, number)	enable a specific pick device
DISABLE DEVICE (pick, number)	disable a specific pick device
SET PICK (number, aperture)	set a square pick window with sides equal to aperture

of devices, and the setting of the pick aperture. Before interrupts from a pick device can be enabled, the pick must be initialized. Commands for a single pick device are listed in Table 2.

When a pick device causes an interrupt, the device number, the segment name of the picked primitive, and the PICK ID of the primitive are placed in an event queue. The GSPC Core proposal specifies that the picked primitive must intersect the pick aperture, but it does not specify the identification algorithm to be used to determine the intersection.

The program can wait for a pick with the command AWAIT EVENT (time, device class, device number). The programmer specifies the time parameter, and the system returns the device class and number. The linking information (tags) in the queue are retrieved by using the command GET PICK DATA (segment name, pick id).

The resulting action returns the segment name and PICK ID for the picked primitive. Note that only one segment name and PICK ID are placed in the queue for each pick interrupt. There is no indication of how many pick tags are in the queue. Thus the program must issue the GET PICK DATA command in a loop to retrieve all the tags from the queue.

The program can empty the event queue for a single pick device with the command FLUSH DEVICE EVENTS (pick, number). This command removes all interrupts for a specific pick.

The programmer may associate a pick with locators, valuators, buttons, and keyboards. *Association* means that when a pick occurs, the associated device is sampled, and the sampled values and the device identifier are placed in the event queue for the device. This data is accessed using commands similar to the pick commands for the associated device (for example, GET LOCATOR DATA). Thus association expands the information available to the program when a pick occurs. The command for association is ASSOCIATE (pick, pick number, locator/valuator, locator/valuator number). There are complementary commands for disassociating devices.

All input devices, including picks, can have an *echo*. An echo is feedback to the user, such as the highlighting of a picked item. Commands are included for specifying, enabling, and disabling echoes for devices.

To summarize, the GSPC Core proposal recommends the following support for picks:

- A static PICK ID attribute (integer valued) for output primitives.
- Dynamic *detectability* and *visibility* attributes for segments.
- Initialization, termination, enabling, and disabling of one or more pick devices.
- Timed waiting for a pick.
- Retrieving the segment name and PICK ID of picked primitives.
- Associating a pick device with a locator or valuator.
- Specifying, enabling, and disabling a pick echo.

This level of support is typical of that provided by many graphical software packages. A common additional function is provision for a vector that indicates the hierarchical structure of the picked item. Some packages also provide a function for retrieving tags of all primitives inside a pick window. If the GSPC Core proposal were extended to include the nesting of segments, these functions could be included by allowing the GET PICK DATA command to return a vector of segment names and PICK IDs for each item inside the pick window.

Requirements for high-level pick support

Software support for picks, as described in the GSPC Core proposal, would have to be augmented to support such applications as circuit design, map digitization, business applications, and spatial layout. Based on experience with applications of these types,[3–7] we have identified six requirements for high-level pick support, which could be built on top of software, such as that specified by the GSPC Core proposal.

specifying the pick window

In most graphics systems, the programmer has no control over the size or shape of the pick window. Because graphical applications create pictures of different densities, and because users have different levels of skill in using pick devices, it is useful to control the specification of the pick window. For example, a street map with 2000 lines requires a finer pick than one with 50 lines. This requirement can be met by allowing dynamic control of the size and shape of the pick window. Pick aperture control, as described in the GSPC Core proposal, is an example of basic control of pick window size. However, rectangular windows and windows that vary with the primitive or segment are also useful.

Most graphics systems require the pick window to intersect the item being picked. Yet in many applications the user may want to pick by pointing inside the item. For example, in revising an office layout it is natural to point inside the graphical symbol to select the piece of furniture to be moved. This capability cannot be supported by such pick devices as light pens, which require the user to point at the item. However, control over pick size or a pick-inside-of function can provide the required function for other pick devices.

dynamic control over detectability

Some graphics applications, such as those for constructing time-series graphs of financial data, may require that during interaction, specific lines or points be made detectable or not detectable. Dynamic control over detectability can be supported by segmenting the picture and using the segment attribute DETECTABILITY, as proposed by the GSPC. The picture, however, must be segmented according to the detectability of items, which may require a large number of otherwise meaningless segments.

A better way of achieving dynamic detectability is to add a WHERE (condition) clause to the SET DETECTABILITY command of the GSPC Core proposal. Using this command, dynamic detectability can be achieved without artificially segmenting pictures. Alternatively, one can modify the GET PICK DATA command to include a WHERE (condition) clause to allow the programmer to accept picks only for items of interest without changing detectability. Note that these WHERE (condition) clauses can be handled as queries by a database management system, as discussed

later in this paper. As an example of a WHERE clause, consider GET PICK DATA WHERE (COLOR = RED). The attributes of the picked items are checked so that only the red items are returned.

resolving pick ambiguities

Pick ambiguity refers to the case when more than one item lies within a pick window, and often arises when one picks near intersecting lines or among items that are close together. Most graphics systems return only one tag when a pick occurs and do not even indicate that more than one item is in the pick window. This limitation forces the programmer to understand the pick identification algorithm. Thus the programmer may have to draw all the points in a picture before drawing any lines, to make it possible for a user to pick a point in preference to a line. Even if the tags of all items in the pick window were returned, there would still be the problem of determining the user's intention.

The programmer may segment the picture to avoid ambiguities. For example, all the points may go in one segment and all the lines in another. Alternatively, in a GSPC Core system, if each primitive were put in a separate segment, segments with points could be given a higher detectability priority than segments with lines. This, however, would require an artificial segmentation that might conflict with the segmentation used to specify detectability. Alternatively, the programmer could resolve ambiguities by writing code or finding clever ways of encoding pick tags. Even worse, the programmer could throw the responsibility back to the users, forcing them to pick (with feedback) until the desired item was chosen.

Dynamic control of pick window size can help resolve pick ambiguities. Also, if the tags of all the picked items are returned, a GET PICK DATA WHERE function and a GET NEXT PICK DATA function are useful in resolving pick ambiguities because they permit retrieval of specific pick tags.

meaningful tags

When one picks an item, the application program may perform the linking function, that is, link the picked item with data or programs. For example, if a user picks a menu item, the application program links the picked command with the name of the program to be executed. Similarly, if a user picks a street to retrieve its width, the application program links the street picked with a key to retrieve the data.

Linking is simpler if pick tags are meaningful values instead of arbitrary numbers. A meaningful value may be a data base key, a program name, or the name for some data. With meaningful values, the application programmer can use the pick tag directly to execute a program or to retrieve data rather than having to decode the tag to determine what to execute or retrieve.

structure tags

Closely related to the need for meaningful tags is the need for tags to reflect the hierarchical structure of the picked item. For example, in a circuit design application when the user picks a line in a resistor, the application program may need to know the vector of tags for the line that is part of a resistor that is in turn part of a component. The programmer may also want to specify the depth of the structure associated with a tag when it is created and the depth of the structure returned when a pick occurs.

Most graphics systems do not provide the structure information as part of the tag. Those that do usually do not give the application programmer control over the depth of the structured tag when it is created or the depth of the tag returned as a result of a pick.

tag data management

A graphics application containing 20 pictures, each with 1000 detectable items, would require 20 000 pick tags. Perhaps only 1000 of these would be active at any one time, but they would all have to be created and stored. If each tag were to take two bytes, 40 000 bytes of storage would be required for pick tags. If meaningful tags and structured tags were supported, the storage requirements could easily exceed 100 000 bytes. In addition, if the x-y coordinates of every detectable item were stored in a pick table, the storage requirements could double or even triple.

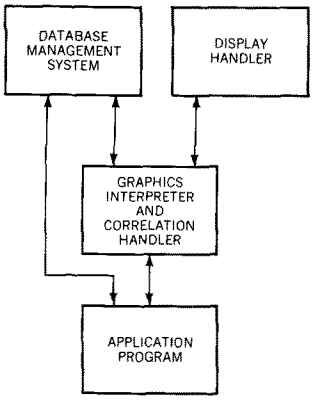
Since all this data cannot normally fit in high-speed storage, there is a need to move the data between high- and low-speed memory. Also, as mentioned previously, the application programmer may require selective access to the pick data using commands such as GET PICK DATA WHERE or SET DETECTABILITY WHERE. The responsibility for managing tag data and providing selective access to pick data should be the responsibility of the graphics system and not the application programmer. Thus the graphics system requires the normal functions of a data base management system to manage both tags and structured display files.[13]

summary of requirements for pick support

Interactive applications require high-level support for the identification and the linking functions for picks. Most existing graphics systems provide low-level identification functions. The required support for picks can be achieved by combining the identification functions with user-supplied linking functions and a database management system using a general-purpose programming language. Thus we have identified the high-level support for picks that should be included in graphics systems. Six features that would reduce the programming effort required for using picks and would help improve the user interface in interactive applications are the following:

- Program control over the pick window.
- Dynamic control over detectability.

Figure 1 Overview of graphics software architecture



- Help in resolving pick ambiguities.
- Meaningful pick tags.
- Structured tags.
- Storage management and query for pick tags.

All these features can be provided by using a database management system as discussed in the following sections.

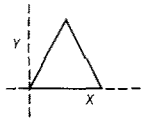
An architecture for high-level pick support

This section describes an architecture for a graphics software system with high-level pick support, such as is shown in Figure 1. The system architecture consists of three major components: a database management system, a graphics interpreter, and a display handler. A prototype based on this architecture is the Picture Building System (PBS).[8,10,11]

database management system

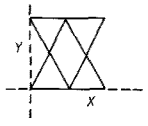
A database system that provides graphical data types is an extension of the concept of a structured display file that is used to store graphical data.[13] Data in the database can have graphical or nongraphical meaning (i.e., semantics). Graphical semantics can be stored as attributes of the data, thereby defining graphical data types. For example, the graphical data type called "coordinate" can be defined as an integer, which is a parameter for draw or move operations, as indicated in Figure 2. If the database contains only graphical data, the database support is similar to a structured display file with data manipulation functions.

Figure 2 Graphics data structuring using a hierarchy of relations



TRIANGLE

X	Y	XYZOPERATION
0	0	MOVE
2	0	LINE
1	2	LINE
0	0	LINE



SHAPE

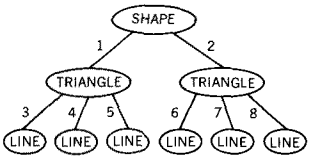
RELATION	SHIFTX	SHIFTY	ROTATION
TRIANGLE	0	0	0
TRIANGLE	2	2	180

User response requirements impose performance constraints on a database management system when the system is used in place of a structured display file. For example, to display a 2000-line drawing in two seconds would require average access times of less than 1 millisecond real time per line. In addition, the database management system should be able to alter data structures in real time in response to user actions. For example, the interactive editing of the picture in Figure 2 might require the addition of columns to add new attributes, the addition of rows to add new items to the picture, and the update of rows to modify the picture.

If a database management system is included in a graphics system, a richer level of pick support can be provided. All graphical data can be stored in the database and the pick tags can be the associated database keys. The linking function for picks thus becomes retrieval by key from the database. In a relational (or tabular) database,[16–18] the basic item that can be picked is a tuple (row), which may represent, among other things, a line, a text string, or a call to another relation. A displayed relation may be represented by a tree, as in Figure 3. Here, the internal nodes represent relations, the branches represent tuples, and the leaves represent display items.

Consider the relation SHAPE in Figure 2. Suppose one wishes to pick the first line in relation TRIANGLE, as called from the first tuple of SHAPE. Returning a single key for this tuple in TRIANGLE is not sufficient, since TRIANGLE is called twice from SHAPE. In general, a vector of keys for an entire path down the tree is necessary to determine the access path for an item.

Figure 3 Correlation table



TYPE	X1	X2	Y1	Y2	J
LINE	0	2	2	2	(2, 3)
LINE	2	2	1	0	(2, 4)
LINE	1	0	0	2	(2, 5)

To indicate that an item is detectable, a pick column can be added to a relation. This column can be a single bit that indicates the detectability of the item and of any associated substructure. In Figure 2, the pick column indicates that one triangle is detectable (PICK = 1) and the other is not. In the detectable triangle, all its lines are detectable because the pick attribute is inherited. By setting the pick column "on" or "off" and displaying the picture again, any subset of the picture can be made detectable. The commands needed for resetting the pick column are queries against the database. For example, in a relation with columns for color and price, it would be possible to make detectable only red items that cost more than 26 dollars.

graphical interpreter

The graphical interpreter is used to produce graphical output from the database and to handle graphical input. To produce graphical output, the interpreter reads data from the database, thereby producing commands to the display handler based on the syntax and semantics of the data. To handle graphical input, the interpreter produces commands to the display handler to perform the identification function. The interpretation of the RELATION column of the SHAPE relation of Figure 2 causes the interpretation of the TRIANGLE relation. This interpretation produces a series of MOVE and LINE commands to the display handler with appropriate shift, rotation, and pick tags. After the interpretation of the output is complete, the interpreter enables the pick device. When a pick occurs, the interpreter performs the identification and linking functions.

display handler

The display handler may be a graphics subroutine package such as those surveyed by the GSPC[12] or the GSPC Core proposal interface itself.[1] It may also be a data communications package such as IBM Message Format Service[19] or simply a hardware interpreter of device order codes. The display handler is driven by commands from the graphical interpreter, or possibly directly from the application program to produce graphical output. To support picks, the display handler must produce and manage a pick table and perform the identification function.

Since pick table management is normally not available in graphics systems, we discuss here the functions needed to manipulate pick tables. Figure 3 shows a pick table and the tree structure of the SHAPE relation in Figure 2. The pick table consists of a row for each detectable item. Each row consists of an item type (line,

Table 3 Functions to manipulate a pick table

ADD—adds a row to the pick table for an item.
DELETE—deletes a row from the pick table (which is useful when an item is deleted from the screen or made undetectable).
FIND FIRST—returns the tag of the first item near the pick.
FIND NEXT—returns the tag of the next item near the pick.
SEARCH DIRECTION—sets pick table search direction to forward (in the same order as drawn) or backward.
SET PICK WINDOW SIZE—sets the tolerance (aperture) for the identification algorithm described in the following section.

text, etc.), its extent, and a vector of tags. The display handler provides the functions in Table 3 to manipulate the pick table. Table 3 shows that the display handler requires a number of database management functions.

application program interface

An *application* consists of application programs and application data. Application data can be stored in relations that can be created interactively. An application program uses commands to the interpreter to handle graphical output and commands to the database system to handle nongraphic data.

To display a graphical object, the application program invokes the interpreter with the command DISPLAY(NAME), in which NAME is the name of a relation in the database. In drawing the picture, the interpreter creates the pick tags for any detectable items. The application program does not have to create or manage the tags. In creating the graphical data, however, the application programmer has to decide which items are detectable.

To use picks, the application program uses a command such as PICK (device name, wait time, tag vector, x, y, button). The device name is needed only if more than one input device is used. When a pick occurs, the application program receives the vector of pick tags, the x-y coordinates of the pick, and the button pressed, if any. Because the interpreter uses database keys as tags, the application program can manipulate the picked item in the database without knowing the tags. Commands such as READP, WRITEP, DELETEP, and DISPLAYP implicitly use the pick tags to read, write, delete, and display data related to the picked item.

The pick device can be considered as a graphical item and thus can be included in a relation just as a line or point. When the interpreter encounters a pick item in the relation, the PICK com-

mand is issued. Note that locators, valuators, buttons, and keyboard can also be treated as graphical items.

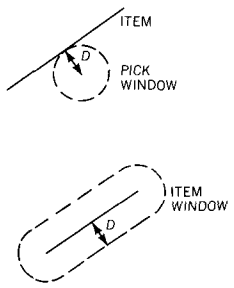
The application programmer need only learn about the detectability attribute (data type) and possibly about a pick item. The database system and interpreter can handle picks in the same manner as any other graphical item. Commands such as READP mentioned previously might be included to provide specialized functions. The pick identification and linking functions are provided by the database system, the interpreter, and the display handler, and are transparent to the application programmer. These functions could be built on the GSPC proposed Core interface to provide the level of pick support discussed here and previously in this paper.

Identification algorithms

Identification algorithms are the procedures needed to determine whether the x and y coordinates of a pick are close to an item on the screen. The identification algorithm must check for intersection of the pick window and each detectable item. The coordinates of the detectable items may be stored in a pick table or recalculated from the graphical data for the picture. We consider in detail the calculations that determine whether a pick window intersects a given item on the screen. (The algorithms extend to three dimensions in a natural way, but such extensions are not discussed here.)

There are two main ways of looking at the intersection of a pick and an item on the screen. One may consider the pick to be a point and each item on the screen to have some window about it (that is, the items are "fuzzy"). Then one detects an item if the pick intersects the item window. On the other hand, one may consider the pick to have a window about it (that is, the pick is "fuzzy"), and each item on the screen is the area of the item. Then one detects an item if the pick window intersects the item.

Figure 4 Equivalent pick and item windows



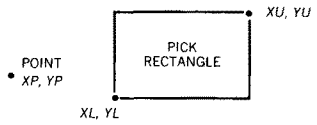
The two points of view are equivalent in function; that is, for any identification algorithm using item windows, there is an equivalent algorithm using pick windows. For example, in Figure 4, an item window of a fixed distance D from any point on the item is equivalent to a circular pick window with radius D. Item and pick windows can both vary with the item, as discussed later in this paper. Since the pick window is simpler to visualize we shall use it in presenting the algorithms; that is, we consider the pick to be "fuzzy."

pick-item intersection

Consider the necessary types of intersections and the corresponding computations. The most general item we discuss is a

closed polygon. This restriction causes no loss of generality, because any item on the screen can be approximated by a polygon. Some possibilities for a pick window are a point, a line, a square, a rectangle, or a polygon. Since the intersection algorithms become increasingly complex as the pick window becomes more complicated, it is advantageous to use the simplest pick window possible. On the other hand, we have found the need for the pick window to be at least a rectangle. Therefore, we choose a rectangular pick window with sides parallel to the coordinate axes.

Figure 5 Point and pick rectangle for the intersection algorithm



We describe algorithms for the intersection of a rectangular pick window with points, lines (covers raster data), rectangles (covers text strings), and polygons. In these algorithms we try to minimize the number of multiplications and divisions. The pick rectangle in Figure 5 is considered to have lower-left-corner coordinates XL, YL and upper-right-corner coordinates XU, YU.

point intersects pick rectangle

To decide whether a point XP, YP lies inside the pick rectangle, one need perform only the following test:

If

$$XL < XP < XU$$

and

$$YL < YP < YU$$

then the point is inside the rectangle.

line intersects pick rectangle

There are many ways of deciding whether a line intersects a rectangle. (See Figure 6.) The goal is to develop a method that is both simple and efficient. We first make the following preliminary check:

If

$$XL1 < XL \text{ and } XL2 < XL$$

or

$$XU < XL1 \text{ and } XU < XL2$$

or

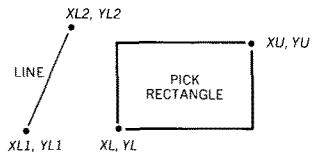
$$YL1 < YL \text{ and } YL2 < YL$$

or

$$YU < YL1 \text{ and } YU < YL2$$

then the line does not intersect the pick rectangle.

Figure 6 Line and pick rectangle for the intersection algorithm



After passing this preliminary test, the line intersects the pick rectangle if two diagonal points of the pick rectangle are on opposite sides of the line, or if the line lies along the boundary of a pick rectangle. We begin with the equation of a line, $y = M*x + B$,

where the slope M and the intercept B are calculated as follows: $M = (YL2 - YL1)/(XL2 - XL1)$ and $B = YL1 - M*L1$. Let $z = y - M*x - B$. Then the point x, y is on the line if $z = 0$; below the line if the sign of z is negative; and above the line if the sign of z is positive. Let

$$XLL = YL - M*XL - B$$

$$ZUR = YU - M*XU - B$$

$$ZUL = YU - M*XL - B$$

$$ZLR = YL - M*XU - B$$

Thus the following test determines whether two diagonal points of the pick rectangle are on opposite sides of the line:

If

$$\text{sign}(ZLL) \neq \text{sign}(ZUR)$$

or

$$\text{sign}(ZUL) \neq \text{sign}(ZLR)$$

or

$$ZLL = 0 \text{ or } ZUR = 0$$

or

$$ZUL = 0 \text{ or } ZLR = 0$$

then the line intersects the pick rectangle.

polygon intersects pick rectangle

When a polygon intersects the pick rectangle, the algorithm consists of two steps. The first is to determine whether any of the lines of the polygon intersect the pick rectangle by repeated application of the algorithm for line intersections with the pick rectangle. If there are any intersections, the polygon does intersect the pick rectangle. If there are no intersections, the pick rectangle lies either completely inside or completely outside the polygon and step two must be performed to determine which condition exists.

The second step is to determine whether the center of the pick rectangle is inside the polygon. The algorithm used is known as the *point-in-polygon algorithm*,[20] and consists of counting the number of times an infinite ray in some direction from the pick window center crosses the polygon. If the number of crossings is odd, the point is inside the polygon, and the polygon does intersect the pick rectangle. If the number of crossings is even, the polygon does not intersect the rectangle. The calculations for these intersections are similar to those that apply when a line intersects the pick rectangle, as just described. Selecting the ray so that it does not intersect any of the vertices of the polygon avoids ambiguity in counting intersections.

The same two-step algorithm is used when the pick rectangle is intersected by a rectangle, which is considered a special case of a polygon.

This section has presented simple, relatively efficient identification algorithms that are part of the display handler. The algorithms are activated by the PICK command, and the pick window is set by the SET PICK WINDOW SIZE command.

Correlation data management

In this section we show how a database management system helps solve the problems and meet the requirements of high-level pick support. The database management system is used to manipulate two types of data: graphical information and pick tables.

dynamic control of pick window

If the graphical information for each item on the screen is stored in a database, the pick tolerance can be included for each item to control its pick window. For example, consider the SHAPE table in Figure 2. Columns that specify X and Y pick tolerance can be added to the SHAPE relation (table). This addition allows a different window for each item.

dynamic control of pick attribute

If a number of pick tables and the graphical information for each item are stored in the database, the value of the pick column (as in the above example) specifies a pick table number or name. One may then issue pick commands based on different pick tables to activate and deactivate the detectability of items. Note that the control of detectability is independent of the structure or segmentation of the picture.

resolving pick ambiguities

If the graphical information for each item is stored in a data base, the data base can be queried for information about any picked item to help resolve pick ambiguities. As an example, again consider the SHAPE table in Figure 2, with both triangles detectable. Suppose one wants to pick the triangle with no rotation. Then after a pick one can issue a READP command to read the ROTATION of the picked item and accept the picked item only if its ROTATION is zero.

meaningful pick tags

If the graphical information for each item is stored in a database, the graphical interpreter can use database keys for tags so that the application programmer need never be concerned with defining or managing tags. The pick table in Figure 3 illustrates this method of handling tags. Using meaningful tags also allows the graphics system to perform the linking function. An application programmer may directly control tags by adding a column specifying a tag to the graphical information.

If the graphical information for each item is stored in a database, the problem of identifying the structure of a picked item is solved by using structured tags (with database keys as values) having a structure identical to that of the item in the database. As an example consider the pick table in Figure 3, where the vector of tags reflects exactly the tree structure of SHAPE.

structure of picked item

The problem created by a large number of detectable items is solved by having the pick table or tables stored in the database. Since database systems can handle tables with a variable number of rows, pick table management is of no concern to the application programmer. Problems of paging and compacting fragmented storage are handled by the database system. Note that this solution places performance requirements on the database management system.

handling a large number of items

Summary

Software support for graphical interaction has been discussed in this paper. Existing support, including the GSPC Core proposal for graphical interaction, has been examined and a list of requirements for high-level support given. The architecture of a prototype system for supporting graphical interaction has been described. This support could be built on top of a device-independent interface, such as the GSPC Core proposal. Algorithms have been presented for identifying the item that a user has selected from the screen. Finally we have shown how the inclusion of a database management system as part of graphical software support can help meet the requirements of interactive graphics application programs.

CITED REFERENCES

1. "Status Report of the Graphics Standards Committee of ACM/SIGGRAPH," *Computer Graphics* **13,** No. 3, I1-V10 (Fall 1979).
2. J. D. Foley and V. L. Wallace, "The art of natural man-machine conversation," *Proceedings of the IEEE* **62,** No. 4, 462-471 (April 1974).
3. J. L. Bennett, "User oriented graphics systems for decision support in unstructured tasks," *User Oriented Design of Interactive Graphics Systems* (S. Treu, ed.), Association for Computing Machinery, 1133 Avenue of the Americas, New York (1977), pp. 3-11.
4. E. D. Carlson, "Graphics terminal requirements for the 1970's," *Computer* **9,** No. 8, 37-45 (August 1976).
5. *Trend Analysis/370 General Information Manual*, IBM Systems Library, order number GH20-1861, available through IBM branch offices.
6. P. Reisner, *Using a Formal Grammar in Human Factors Design of an Interactive Graphics System*, Research Report RJ 2505, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (April 1979).
7. J. P. Jacob, *Potential of Graphics to Enhance Decision Analysis*, Research Report RJ 2437, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (January 1979).
8. D. Weller and R. Williams, "Graphic and relational data base support for problem solving," *Computer Graphics* **10,** No. 2, 183-189 (Summer 1976).

9. D. Weller and F. P. Palermo, "Database requirements for graphics," *IEEE Computer Society International Conference, 18th, COMPCON '79, Spring, San Francisco, CA, Feb. 26-Mar. 1, 1979*, IEEE Computer Society, Long Beach, CA (1979), pp. 231-234.
10. H. G. Meder and F. P. Palermo, "Data base support and interactive graphics," *Proceedings, Third International Conference on Very Large Data Bases, Tokyo, Japan, Oct. 6-8, (1977)*, IEEE, New York (1977), pp. 396-402.
11. F. P. Palermo and D. Weller, "Picture building system," *IEEE Computer Society International Conference, 18th, COMPCON '79, Spring, San Francisco, CA, Feb. 26-Mar. 1, 1979*, IEEE Computer Society, Long Beach, CA (1979), pp. 235-237.
12. "Status Report of the Graphics Standards Committee of ACM/SIGGRAPH," *Computer Graphics* **11,** No. 3, I1-I19, II1-II117 (Fall 1977).
13. W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill Book Company, Inc., New York (1973).
14. D. Bantz, "Proposal for a display processor," *IEEE Transactions on Computers* **C-17,** 54-55 (1968).
15. I. E. Sutherland, "Sketchpad: a man-machine graphical communication system," *Proceedings of the Spring Joint Computer Conference*, Spartan Books, Baltimore, MD (1963), pp. 329-346.
16. E. F. Codd, "A relational model for data for large shared data banks," *Communications of the ACM* **13,** No. 6, 377-387 (June 1970).
17. D. D. Chamberlin, "Relational data base management systems," *Computing Surveys* **8,** No. 1, 43-66 (March 1976).
18. R. D. Bergeron, P. R. Bono, and J. D. Foley, "Graphics programming using the Core System," *Computing Surveys* **10,** No. 4, 389-443 (December 1978).
19. *IMS/VS Message Format Service User's Guide*, IBM Systems Library, order number SH20-9053, available through IBM branch offices.
20. J. D. Jacobsen, "Geometric relationships for retrieval of geographic information," *IBM Systems Journal* **7,** Nos. 3 and 4, 331-341 (1968).

D. L. Weller, E. D. Carlson, R. Williams, and S. N. Zilles are located at the IBM Research Laboratory, 5600 Cottle Road, San Jose, CA 95193. G. M. Giddings is located at DiscoVision Associates, 3300 Hyland Avenue, Costa Mesa, CA 92626. F. P. Palermo is located at the IBM Santa Teresa Laboratory, 555 Bailey Avenue, P.O. Box 50020, San Jose, CA 95150.

Reprint Order No. G321-5127.