Network Working Group
Request for Comments: 192
NIC: 7137

R. Watson
SRI-ARC

12 July 1971

Some Factors which a Network Graphics Protocol must Consider

After reading some of the RFC's on a network graphics protocol it seems that many are not providing general enough mechanisms to handle attention handling, picture structure, and other higher level processes involved in interactive graphics.

Therefore for what it is worth I am sending out these rough introductory notes which contain ideas that I think any network graphics protocol must come to grips with.

The network graphics protocol should allow one to operate the most sophisticated system with more general data structures and concepts than those described in these notes and allow very simple systems to function also.

Introduction

It is our contention that, if computer graphics is to be widely useful, the graphics terminals must be just another type of terminal on a timesharing system with minimal special privileges. In these brief notes we outline the basic features which we feel must be available in a graphics support package to allow easy interactive graphics application programming.

If one examines the types of tasks in industry, government and universities which can avail themselves of timesharing support from graphics consoles, one can estimate that the large majority can effectively utilize quite simple terminals such as those employing storage tubes. I would estimate 75% of the required terminals to fall in this class. Another 15-20% of terminals may require higher response and some simple realtime picture movement, thus requiring simple refresh displays. The remainder of terminals are needed for high payout tasks requiring all the picture processing power one can make available. In this talk we are not considering support for this latter class of applications.

MAIN ASSUMPTIONS AND REQUIREMENTS FOR SYSTEM DESIGN

The main assumptions and requirements underlying the interactive graphics are the following:

1) The user of the graphics terminal should be just another timesharing system user.

2) The graphics software support should interface to existing timesharing programs.

3) The software support should allow technicians, engineers, scientist, and business analysts as well as professional programmers to easily create applications using a graphic terminal.

4) The software support should easily allow use of new terminals and types of terminals as they come on the market.

5) The software support should be expandable as experience indicates new facilities are required.

6) The software support should be portable from one timesharing service to another.

7) Some form of hardcopy should be available.

MULTILEVEL MODULAR APPROACH TO SYSTEM DESIGN

If one wants to create as system which is easy to use by inexperienced programmers and ultimately non-programmers, one needs to provide powerful problem-oriented aids to program writing. One has to start with the primitive machine language used to command the graphics system hardware and build upward. The philosophy of design chosen is the one becoming more common in the computer industry, which is to design increasingly more powerful levels of programming support, each of which interfaces to its surrounding levels and builds on the lower levels. It is important to try to design these levels more or less at the same time so that the experience with each will feed back on the designs of the others before they are frozen and difficult to change.

One can recognize five basic levels:

1) The basic system level:

   This level provides facilities for use of the terminal by the assembly language programmers.

2) The problem programming language level:

This level of support provides powerful facilities for interactive graphics programming from the commonly used higher level programming languages.

3) The picture editor or drawing system:

This level of support allows pictures to be drawn and linkage to these pictures and application programs.

Data management support for interactive programming:

This level of support is to provide facilities to aid creation and manipulation of data structures relating data associated with the pictures and the application.

5) The application program level:

A REVIEW OF TERMINAL HARDWARE CHARACTERISTICS OF CONCERN TO THE USERS

There are two basic kinds of general purpose cathode ray tube display systems available on the present market. Within each class there are alternate forms and techniques of implementation which we do not discuss here. One type is called a "refresh display". The other type is called a "storage tube display". The refresh display must keep repainting the picture on the screen at rates of from 20-60 times per second. Commands which instruct the system how to draw the picture are stored in a memory. The storage tube display on the other hand, through its internal method of construction can maintain on the face of the display a picture for practical purposes, indefinitely once drawn.

REFRESHED DISPLAYS

There are limits to how much information can be drawn on the face of refreshed display before the time required to paint it forces the refresh rate below a critical value and the picture appears to flicker. This quantity of information is a function of the type of phosphor on the tube face, the speed of display system in drawing lines and characters, and the ambient light level in the room. Refresh display systems range in cost upwards from $10,000 to several hundred thousand dollars. Refresh displays, because the picture can be changed every few milliseconds by simply altering its command list (often called a display file or display buffer), allow the picture parts to be moved on the face of the screen either under operator control or computer control. Objects on the screen can be selectively erased without affecting other objects on the screen.

These characteristics make refreshed displays suitable for a wide range of applications.

STORAGE TUBE DISPLAYS

Storage tube based displays can display a large amount of information without a flicker, and generally cost under $20,000. Present systems suffer from some limitations, however. They cannot be selectively erased. If an object is to be moved or deleted from the screen, the entire screen must be erased and then the new picture can be redrawn. Because this type of display generally operates over a communication line, the speed of the line may seriously restrict the amount of interaction if much erasing and redrawing is required. The graphics software concepts to be described can be used with both a storage tube and refreshed display, although some features are only appropriate to the refreshed type of display. The important point is that new storage tube technologies insure that this class of terminal will be with us a long time.

INPUT DEVICES

It is necessary to allow a console user to communicate with the graphics system. This is done through a keyboard and through specialized graphic input devices, the Light Pen, the Tablet, the SRI "Mouse", and the "Joy Stick". These latter devices enable a console user to point to vectors and characters displayed on the CRT and to input position information to the graphics system.

Comparison of the Graphics Input Devices -- Analog Comparitors

The Joy Stick, Mouse, and Tablet are similar in that they both generate a two dimensional position address without the aid of the display processor, but cannot be directly used to identify displayed objects. The light pen-display processor hardware combination and its associated software, on the other hand, can easily sense and identify displayed vectors and characters but does not generate directly any position data. A "tracking cross" program is used to obtain the position data for the light pen. To obtain the pointing capability for the Joy Stick, Mouse, and Tablet, we can use a pair of analog comparitors which generate interrupts when the beam is drawn on the CRT lies within a rectangular "viewing window" in much the same way that the light

pen generates interrupts when a beam is drawn under its circular viewing area. These comparitors sense the x and y axis drive voltages of the display analog bus.

A comparator will generate an output signal when the drive voltage is between two limits which may be set using special display processor commands. When both comparitors generate a signal simultaneously, the output voltages on the analog buss correspond to a beam position within the rectangular viewing window. The position of viewing window is set based on the position of the pen, Mouse, or Joy Stick.

We can also use software to simulate the effect of hardware comparators. Hardware comparators cannot be use with storage tube displays and, therefore, a software simulation is required. This simulation is discussed later in these notes.

The light pen can be used only with a refreshed display. The other types of devices can be used with present storage tube displays and refreshed displays. They are used with storage tube displays which have hardware which produces on the screen a dot, cross or other cursor, indicating the x, y position of the device. The reason one can move this cursor around it that the cursor is created using special techniques to avoid its storing on the screen.

USER SOFTWARE REQUIREMENTS

The user requirements on a timesharing system based interactive graphics system are the following:

1) The user should have available a language for creating a computer representation of the picture to be displayed. This language should allow more complex pictures to be built up from simpler structures.

2) The computer representation of the picture must allow easy identification of picture parts when pointed at or "picked" or "hit" with graphical input devices such as light pen, electronic pen-tablet, Joy Stick, SRI mouse, or other supplying x, y information.

3) The computer representation of the picture must allow linking of picture parts with data about these parts appropriate to the application using the terminal. There should be an appropriate data management system for use with interactive application programming.

4) There must be some way of communicating events taking place at the terminal in real-time, such as picking objects with the light pen, with the application program running in the timesharing system.

5) The user should be able to save and restore pictures from one console session to the next.

6) If possible, the user should be able to use the display as a stand-alone terminal or in conjunction with a teletype or other typewriter terminal.

7) The user should be able to do some graphic programming by drawing directly at the console.

The choice of an appropriate data structure for picture representation simplifies the handling of requirements one to five. It is this data structure that we consider now in more detail.

Picture-Related Structures

If a picture displayed on the console had meaning only in the physical position of its lines and characters, the system would be little more effective than an easily erased piece of paper. To significantly enhance the capabilities of the system, we must be able to express relations between displayed entities. A line is much more than just a line when it represents a boundary or a part of some more complex unit. Such units in turn may be related in a similar way to higher level units. Furthermore, we may wish to create picture elements that may be used repeatedly so that a change in the one master copy will be reflected in every use of that copy.

To illustrate the usefulness of this picture-subpicture relationship, we shall consider the three houses of Figure 1. While the two types of houses differ in appearance, it is obvious that they have picture elements that could be drawn by a designer of prefabricated houses and that the designer wished to incorporate a new standard window unit into all houses. The use of conventional pencil and paper techniques would require that he redraw or overlay each window on his diagram to reflect the changed component. If the window were, instead, drawn by the graphics system within a common subroutine, only that one master copy would have to be modified in order to change the appearance of every reference to that kind of window on the diagram.

Nodes and Branches

To facilitate the discussion we will introduce the terms "node" and "branch". A node is a form of picture subroutine that may cause the display of lines and characters and may also call other nodes. The subroutine call is called a "branch". Nodes may also be thought of as representing PICTURES or SUBPICTURES and the branches to these nodes as uses or instances of these subpictures.

Directed Graph Structure

The nodes and branches form a directed graph. The branches contain positioning information indicating the beam location to be used by the called node. This location is relative to the position of the node in which the branch is made. This use of relative beam positions allows the user of the system to create subroutine structures that make multiple branches to common nodes. Branches may also set other display parameters such as intensity and character size. A subroutine calling structure appropriate to the requirements of our hypothetical designer is shown schematically in Figure 2. Nodes are shown as circles and branches are shown as connecting lines. The picture of the house is composed of wall unit and roof SUBPICTURES. The wall unit is in turn composed of subpictures.

Node and Branch Display Parameters

Branches may contain the setting of parameters which will be in effect when the called node is executed. The parameters which may be set are the beam position to be used (relative to the current beam position, i.e., a displacement value), intensity, character size, line type, visibility, (the display of vectors and characters may be suppressed), "hitablility" (whether or not vectors and text may be "viewed" by devices such as the light pen), and blinking.

Coding within nodes may modify only the parameters controlling position, intensity, character size, and line type to be used by subsequent display coding or branches. It is not necessary that a node or branch specify every parameter. For those parameters other than position, the system allows a "don’t care" option; the parameter setting in effect when the node or branch is executed will be retained and used in this case.

Identification of Graphic Entities with Graphic Input Devices

Structural Hits

A console operator or application program may modify, add, or delete branches to any of the nodes as well as add new nodes. To allow a console operator to manipulate any branch in such a structure, we have implemented a "structural hit identification" scheme. To illustrate the following discussion, we refer the reader to Figures 1 and 2.

A viewing device, such as a light pen, can respond only to the individual vectors or characters displayed on the screen. At the time a vector is drawn under the viewing area of the light pen, an interrupt is generated and, if enabled, will be sent to the central computer. Even though the same node is used to display the eight windows in the diagram of Figure 1, we can tell which window and house is being pointed to by examining the sequence of branches taken to arrive at the window displayed at the time of interrupt. If the console user points to the right hand window of the middle house of Figure 1 (marked with an asterisk *) an examination of the subroutine return addresses in the push down stack would show that the current "window" node had been arrived at via the dotted line path shown on the network of Figure 2.

There remains the question "Are we pointing at a window, at a wall, at the house, or at all three houses?" The location of this structural hit depends on how many branches are counted in examination of the return addresses before one stops to consider to which branch that return jump points. This is analogous to counting a fixed number of levels from the ends of the graph structure. This number of jumps is set using reserved keys on the keyboard, one incrementing and the other decrementing the limit. By manipulating these keys and pointing to various displayed objects with the light pen, it is possible to point to any branch in the network of subroutine calls.

All information concerning the path in the node-branch network taken to arrive at any displayable coding is contained in a push down stack. Return jumps are stored in the stack by the subroutine calls to nodes. These jumps when executed will return the processor to the next instruction after the call.

A greatly simplified version of the display coding used to generate the picture and tree of Figures 1 and 2 is shown in Figure 3. The labels a through d on the diagram represent the

address of the subroutine calls which cause the display of the subpicture hit by the viewing device -- in this case the right hand window of the second house. The returns from the called subroutines are stored in the push down stack as jumps to the location following the calls. The routine RETURN would merely execute POP instructions which ultimately will cause the execution of a jump instruction previously placed in the stack by the calling branch, thus returning control to the calling routine. The stack is shown in the condition at the time of the hit on the right hand window of the middle house. Note that by counting 3 jumps upward (downward in the diagram) in the memory containing the stack, we will arrive at the jump pointing to a structural hit at (b) in Figure 3, the call to model 120.

Console Operator Feedback

The console operator must be informed of where he is pointing in the network of nodes and branches. This is accomplished by flashing all displayable coding below the structurally hit branch when a vector or character is viewed. This flashing is a doubling of the intensity at 2 to 8 cycles per second. In addition, a list of the names of all nodes and branches taken to arrive at the vector or character viewed is displayed in a corner of the screen. The name of the branch selected is intensified somewhat brighter than the other names.

Generating an Attention

After the operator has confirmed the correctness of his choice, he need only terminate the view in order to generate an attention on the desired branch. This is done by releasing the button on the light pen or lifting the pen from the Tablet. A button on the mouse will perform the same function. If the structural hit is not correct then the operator could move the viewing device to a new area.

A termination of the view on a blank area of the screen will result in the generation of a "null" attention. This attention returns only position data; no structural data is generated. The significance of this attention is determined by the application program.

The above discussion assumed a refreshed display and use of a light pen, but it greatly simplifies interactive graphics programming if the above concepts can be implemented no matter what type of display or graphical input device is being used. This in fact can be accomplished as discussed later.

THE GRAPHICS LANGUAGE

For the purpose of discussion we assume that the graphics language statements are a set of subroutine calls, although a more sophisticated syntax could be imbedded in the host programming language. The statements required are:

1) Subroutine calls for creation and manipulation of the picture-subpicture data structure.

2) Subroutine calls to generate displayed pictures and picture parts such as lines and characters.

3) Subroutine calls to input information about events or "attentions" occurring in real time at the console.

4) Subroutine calls to manipulate picture parameters such as line type, (solid, dashed, dotted, etc.), brightness, character size, and so forth.

5) Subroutine calls to perform utility functions such as saving and restoring pictures from disk files, initiating the display and so forth.

NAMING

A number of different naming conventions are required to meet system and application programmer needs.

The Display Pointer

Nodes and branches in the system are named by assigning an integer or array location as an argument in the call used to create them. The system places in these variables a number which points to the physical location of the branch or node position in the picture-subpicture data structure. We call this name the DISPLAY POINTER. As long as the user does not change the contents of these variables he can refer to particular nodes or branches in various subroutines by use of these integer variables as arguments. In other words, to the user, the name of a picture or subpicture can be thought of as the variable used at the time of its creation. Such a naming scheme is clearly required if pictures or subpictures are to be manipulated by the programmer.

The Light Button Code

Additional identification is useful to the application programmer in order to simplify his programming task. A user has no control over the number assigned by the system to a Display Pointer. There are situations in which the user would like to associate a particular known number with a branch. One common example is in the use of "light buttons". A light button is a displayed object that the user wants to be able to point at in order to command the controlling application program to do something. A light button is commonly a string of characters forming an English word or words, but could be any picture. When the user picks or hits the light button, information identifying the object must be transmitted to the timesharing application program. The program must then branch to an appropriate statement or subroutine to perform the operations required to execute the command. The Display Pointer uniquely identifies the object hit, but because its value is not under the programmers control, writing the code necessary to test it against the various Display Pointers considered legitimate to be hit at this point in the program is tedious. If, however, the application programmer knew that at this point only objects with identification numbers 20-28 were legitimate to be hit, then testing to see that one was in this range and branching by use of a computed GOTO simplifies the programming of flow of control. Often one does not need unique identification of an object, but wants to perform a certain action if any object in a class of objects is hit.

The above need for identification is satisfied by allowing the application programmer the ability to assign a number, not necessarily unique, to a branch. This number is called the Light Button Code. This code can be used in any way the programmer desires, but is most commonly used, as its name implies, as a code identifying light buttons. This number is sent to the application program along with the Display pointer of the object hit on the screen with a graphical input device.

The Back Pointer

We indicated earlier that it is required in interactive graphic programming to be able to associate application oriented data with picture and subpicture objects on the screen. The data may be stored in many kinds of data structures depending on the nature of the application, examples being arrays, lists, trees, etc. We meet the need by associating with each branch one word which could contain a pointer to the appropriate spot in the application data structure containing the data associated with

the branch. We call this word the Back Pointer. The application programmer can in fact store any code he desires in this word and use it in any way desired, but its common use as a pointer back into a data base in the application program dictated its name.

For example, consider an application which would allow a chemical engineer to draw a chemical flow sheet on the screen and then input this flow sheet into a process calculation system. There will be various symbol-pictures on the screen representing basic process units such as heat exchangers, mixers, columns, and so forth that can be copied and positioned on the screen. These units will have to be connected together by streams. The units and the streams will have names and data associated with them describing their contents and properties. Further, the node-branch structure. while visually indicating to the user what units are connected together and how, does not necessarily have the connecting information in a form easily handled by the application program.

The continuity is best represented by a data structure using simple list processing in which each unit and stream has a block of cells associated with it containing data for it and pointers containing the connectivity information. When a branch is created to position and display a unit, it will contain in the Back Pointer a pointer to the block of data associated with it. The block of data will probably contain the Display Pointer for the associated branch so that one can go from the picture to the data block or from the data block to the picture. For example, one may point at a unit for the purpose of deleting it. Given the Back Pointer of the unit hit, one can find its associated block and return that block to free space. One can then follow the appropriate chain of pointers to the blocks for the streams connected to the unit. In these blocks one has the Display Pointers for the branches displaying the stream and can then delete it from the node-branch structure, thus making it disappear from the screen.

An additional form of name is to allow the programmer to store an alphanumeric string with each branch or node. This form of name is not required for most applications, but can be useful with the picture editor.

To review, each node and branch has associated with it a unique identifier named by the user and called the Display Pointer; its value is assigned by the system. Each branch has two additional pieces of information which can be assigned to it by the programmer, called the Light Button Code and Back Pointer.

Given a Display Pointer for a branch, the programmer can obtain the Light Button Code or the Back Pointer for the branch. Given a Light Button Code or the Back Pointer, the programmer can obtain a Display Pointer for a branch with such a code. This display pointer may not be unique if several branches have the same Light Button Code or Back Pointer. The above naming and identification inventions have proven to be easy to understand and yet completely general and easy to use.

COORDINATE SYSTEMS

We now consider the question of a coordinate system within which to describe picture position. The actual display generation hardware in a terminal has a fixed coordinate system (commonly 1024 by 1024 units on a fixed size screen with the origin 0,0 in the left hand corner or center on the screen). Ultimately, the user wants to work on a virtual screen much larger than the hardware screen and wants to consider the hardware screen as a window that he can move around to view this virtual screen. Further, pictures are to be capable of being constructed out of subpictures as in the example of Figures 1 and 2. To be able to accomplish the latter and allow future expansion to allow the former, the following coordinate system conventions are used.

Each node has its own coordinate system. When a node A is created, the picture-drawing CRT beam is assumed by the programmer to be at the origin of the node's coordinate system. When a node is used within a node B by use of a branch, the positioning of node A is relative to the beam position in the coordinate system of node B. All nodes are positioned relative to each other by x, y positioners in the corresponding branches. When a picture is actually to be displayed, one node is indicated to the system as the initial or Universe Node. This initial node is positioned absolutely on the screen and all other nodes appear relative to this one as specified in the branches pointing to them. This scheme is required to give the flexibility and generality required in the picture-subpicture tree.

Logical Completeness of Operation Set

Throughout the system design one should try to follow the philosophy of incorporating a logically complete and consistent set of operations. In particular, for each call that sets a value there should be another call to fetch the value. That is, for each operation there is an inverse operation whenever it is meaningful to have one. We see a need for a basic system with the calls as primarily primitives. One can incorporate calls that could be created by the programmer from other calls, when it is

felt that usage would warrant the expansion. We would expect a library of higher level routines in the language.

It is beyond the scope of these notes to go into all the calls required except to indicate a few basic ones. For structure creation, one needs to be able to create a node or branch, delete a branch, add a new branch to a node at run time.

One needs to be able to specify beam movements in nodes and place text in nodes with the normal write-format statements of the host programming language. This latter point is very important for easy programming.

One needs to be able to set and test parameters and convert one form of name into others.

We discuss Attention handling in more detail because of its importance in making interactive programming easy.

Attention Handling

The user sitting at the console is operating in real time while the application program is operating in timesharing time. At any point where the user may perform some operation at the console, the application program may not be running. A mechanism must be created to communicate between the user and the application program. The design of this mechanism is very important and must be carefully considered. There are many different operations that one might want to provide the user at the console. A basic mechanism is discussed which will allow others to be added in the future. When the application program gets to a point where it is expecting input from the terminal, it issues a call and passes an array as an argument. The Attention handling mechanism dismisses the program until an event is reported from the console. The information passed back to the application is the type of event which occurred and other relevant information for that event.

On refreshed displays a common input device is the light pen. The light pen has a physical field of view of about a 1/8-1/4 inch circle. The most common use of the light pen is to point at an object to be hit or picked. The logical field of view seen by the user is a branch in the node-branch structure. The picture drawn by the structure below the branch is blinked to give feedback to the user about what object he is going to hit or operate upon. The level in the structure at which the logical view is given can be set under program control or adjusted by the user from the keyboard. When the user obtains feedback indicating the correct object is in view, he then presses a button on the light pen to

generate an Attention. He is said to obtain a "structural bit" at a branch at the level in the node-branch structure set by the application program or by himself. When the hit occurs, appropriate information is then entered into the Attention queue as described below.

The other type of graphical input device commonly in use on both refreshed and non-refreshed displays, such as electronic pen-tablets, Joy Sticks, SRI Mouse, etc., produce x, y position information which is fedback to the screen as some sort of cursor, such as a dot or a cross. It is difficult, if not impossible, without special hardware to provide the kind of feedback possible with the light pen, but structural hits can be generated by the use of special hardware or software. These devices require the application programmer to set the appropriate level for an expected hit.

The level of a structural hit is counted up from the bottom of the node-branch structure. A hit at level 1 is the lowest branch presently in view. A hit at level 0 is a hit on an individual vector or group of characters. Only special programs, such as a picture editor, are likely to obtain hits at level 0.

The Attention type obtained when one gets a structural hit on a branch returns the following information: The information returned in the array is that required by the application program, the Display Pointer, the Light Button Code, and x, y, information. The x, y, information returned is not the absolute x,y pen position because this would not be of use on this type of hit. The x, y information returned is the physical beam position just before execution of the branch which was hit. If one wants the physical location of the node origin to which the hit branch is connected, one executes another call to obtain the branch positioner and adds these values to the corresponding values obtained from the hit. Given the Display Pointer, one can obtain the Back Pointer or other parameter values associated with the given branch call.

The attention type obtained when a hit is generated, but no object is in view, is now discussed. This type of attention is called a null attention. It is used frequently to position objects on the screen. The only information returned in the array is the absolute screen coordinates of the position on the screen of the graphic input device or cursor. This information can be converted into relative information for placement in a branch positioner or for incrementing a branch position when an object is being moved.

Other calls are required to obtain information about other branches which are related to the one hit, and to perform other functions.

STRUCTURAL HITS FOR STORAGE TUBE DISPLAYS

The final topic is to consider how to obtain structural hit information using a storage tube display or device which only gives absolute x, y screen information.

The problem is to take an x, y coordinate pair and determine if the user is or is not pointing at an object on the screen, and if he is, which object. When a hit is generated with the light pen, the display processor halts and the controlling computer can gain access to the return addresses in the push down stack and to the instruction location which generated the line or character causing the hit. Use of the Joy Stick, Mouse, or tablet is completely asynchronous with the display for refresh displays and the hit occurs after the drawing has taken place for storage tube systems.

The brute force approach to the problem would be to simulate execution of the Display Buffer and calculate some measure of distance between every line and the x, y coordinate of the hit. This approach would be too time consuming and is not feasible. A second approach and one commonly used is to have the programmer define a rectangle surrounding each object on the screen. Then one determines which rectangle the cursor was in and that determines the object hit. This approach requires extra effort by the programmer, and only works well if the node-branch structure is one level deep, there are no diagonal lines as nodes, and no objects have overlapping rectangles. These severe restrictions eliminates this approach from serious consideration.

A third approach would be to break the screen into small squares or rectangles of a size such that it is unlikely a line from more than one picture object would pass through the square or rectangle. Then we would record for each square the Display Pointer of the lowest level object branch passing through it. This approach would require considerable system space and would take much time to determine what rectangles each line passed through.

The fourth approach and the one we recommend is to split the screen into horizontal and vertical strips. When the call to DISPLAY is given, the system makes one pass through the node-branch structure and makes a list of the Display Pointers for the lowest branch having a node with a line or character passing through or in each horizontal or vertical strip.

This calculation can be made quickly because the system can easily obtain the start and end points of a line. One then can quickly determine which strips the end points fall in, as well as the intermediate strips crossed. When a hit is generated, the x, y information is converted to horizontal and vertical strip numbers. The Display Pointers for each of these strips are intersected to see if a common Display Pointer exists. If yes, this is the Display Pointer for the object hit. If not, then a null hit is generated. Choice of strip width decreases the probability of multiple hits resulting.

The above process yields the Display Pointer of the lowest branch in the tree in view, but one may want to obtain information about other higher branches in view. This is accomplished by creating, not only the strip lists described, but by parsing the node-branch structure at the same time into a table containing an abbreviated representation of the tree and the screen x, y coordinates existing at each branch. The strip lists do not actually contain Display Pointers, but pointers back into the parsed representations which has the Display Pointer, x, y coordinates, and the structure level for each of the branches. The parsed representation is a linear list of the branches encountered as the program walks through the node-branch graph. Given the hit at the lowest level one can determine all branches passed through from the top node to the hit branch by an upward search of the graph representation.

Every time a branch is deleted or a new branch is added, one needs to modify the screen, modify the representations and the strip lists. For refresh displays, the picture can be changed immediately and the strip lists and representations modified at the time of an attention call. For a storage display, erasing and redrawing the picture on each deletion can be slow, if many deletions are going on, and may be unnecessary.

There are three approaches to performing these functions in storage tube systems:

1) Erase the screen on each deletion and recompute the picture, strip lists and graph representations on each deletion and addition.

2) Keep a list of each Display Buffer change and perform erase if necessary and redraw or make an addition when an attention call is encountered. This is a feasible approach because it is only at this point that the screen and structural hit information need to be up to date.

3) The third is to allow control of screen changes and other updating by special subroutine call. The recommended approach uses a combination of the above. Adding information to the screen should occur at the time of the new branch call. Deletions and modifications of the representation and the strip lists occur only at the time of an attention call. Routines should also be provided to give the programmer control over this redraw mechanism.

   Experience with the above mechanism has shown it to be quite fast and not to noticeably degrade response time. One minor difficulty has been encountered when a horizontal or vertical line of an object is on the borderline of a strip. Sometimes this results in a null hit being generated if the cursor is on the wrong side of the borderline. A check can be made for this condition and audio feedback can be given to the user with the bell in the terminal to indicate a correct or erroneous hit.

INTERFACE TO THE TIMESHARING SYSTEM OF A REMOTE MINICOMPUTER DRIVEN DISPLAY

Although the graphic system is locally controlled by a minicomputer, the user does not have to worry about the mini. Application programs are written for the timesharing computer only. The graphic system as a whole behaves as a terminal of the timesharing computer. This feature is important because no matter how powerful the graphic system is, it must be easy to program and use before useful applications can be implemented.

Because no one wants to operate over a communication line, one needs to compress the information sent to the remote system. This is accomplished by compiling a central node-branch structure in the central computer and only sending minimal character strings to the remote computer representing those subroutines calls that need to be compiled into a Display Buffer in the remote computer for display refresh. In other words, a smaller remote version of the graphics system resides in the remote minicomputer. Simple schemes for coordinating the Display Pointer in the remote and central machine have to be devised.

CONCLUSION

We feel that the above concepts are central to creating an interactive graphics support system for use with a timesharing system. The key concepts are those associated with the node-branch structure and the structured hit. The topics of a picture editor, data management system, and basic level support are also very important, but beyond the scope of this lecture.

Figures 1, 2. and 3, are available in both .PS and .PDF versions.

[This RFC was put into machine readable form for entry]
[into the online RFC archives by Lorrie Shiota, 10/01]