RTMP's copy

DISPLAY STATION CONTROLLER

The present invention relates generally to electronic computer systems, and more specifically to a system and method for controlling graphics displays for computer systems

Graphical user interfaces are becoming increasingly common in the computer industry. These interfaces typically use various types of images and graphics to convey information to users more quickly and clearly than text based interfaces. Many such interfaces allow multiple, often overlapping, windows representing separate applications or separate parts of the same application, allowing a user to monitor the activities of several programs at once.

Driving such graphical interfaces places heavy demands on the processors involved. In addition to the fact that several applications are often running simultaneously, the use of overlapping windows means that display subsystems must constantly monitor those portions of each window which are visible to the user. Manipulation of bit mapped graphics on a high resolution display, often in color, requires a large number of processor cycles even for relatively simple tasks. As a result, high performance processors are required to control complex graphics displays.

Systems which are not specifically designed to support high resolution graphical displays can suffer performance degradation when driving such displays. This becomes especially true when such systems are required to support multiple displays simultaneously. The amount of processing required simply to support basic functions on multiple graphical displays simultaneously can tax the resources of even a high performance processor, leaving little or no

time for the execution of basic processing functions. This greatly degrades the overall performance of such systems.

In order to increase programmer productivity and portability of applications, numerous graphical display interfaces are in the process of being standardized. One increasingly popular graphical display standard is known as X Windows. X Windows has initially been implemented on UNIX (TM) and compatible operating systems. As such, X Windows is designed to be highly portable across computer systems having widely different architectures and different manufacturers.

X Windows is defined so as to be easily implemented over a local area network. This allows applications to execute on one machine, while the graphical user interface runs on a separate, physically remote, machine or terminal. The formal definition of the X Windows system changes from time to time under the control of an association of major computer manufacturers who support the X standard. Literature describing the X Windows system and its implementations is voluminous, and one document which explains the X Windows environment is X Window System, 2d edition, R.W. Scheifler and J. Gettys, Digital Press, 1990.

X Windows works well with a network of full featured, high performance graphical work stations which have their own operating systems and can run applications. This is, in fact, the environment for which it was designed. An alternative system which could make good use of the graphical X Windows interface would be one which has a large central system, and relatively simple graphical terminals connected to the central system over a network. This arrangement concentrates much of the processing power in one location, and allows additional users to be added to the network for a relatively small incremental cost. Once the system is in place, the incremental cost for adding a user is much less

REPLACED SHEETS SG
☑ Description : paged 3 to paged 4
☑ Claim(s) : paged 17 to paged 20
☐ Drawing(s) : paged ..... to paged .....
☐ Abstract : paged ..... to paged .....

than for providing each new user with a fully featured graphical work station.

One important drawback to the use of a central system is that the processing demand on the main central system is quite high due the large number of graphical terminals which are supported. In addition to the burden of executing a larger number of application programs simultaneously, a heavy processing burden must be carried by the central system simply to support the X Windows graphical interface for the terminals. If more than a very small number of graphical terminals are executing simultaneously using the resources of a main central processor, overall performance of such a processor can be severely degraded.

A single central system driving a number of terminals over a network could be a cost effective alternative to a network of fully featured work stations. However, such a system will only be feasible if the needed performance can be achieved. Such a centralized system is quite flexible, since fully featured work stations can be attached to the network as well as graphical terminals.

It would therefore be desirable to provide a system capable of supporting multiple graphic sessions on remote terminals or other devices over a network. It would be further desirable for such a system to be compatible with X Windows or a similar widely available standard interface. It is highly desirable that such a system provide acceptable performance while driving a relatively large number of graphical terminals.

It is therefore an object of the present invention to provide a system for supporting multiple graphics sessions on remote (network attached) terminals.

This Sheet Has Been Replaced

It is another object of the present invention to provide such a system which is compatible with the X Windows interface standard.

It is a further object of the present invention to provide such a system which provides good performance for a relatively large number of multiple, simultaneous remote sessions.

Therefore, according to the present invention, a system for supporting graphics display sessions on remote terminals contains a processing subsystem dedicated to graphics support. Applications execute on a central processor, and display related graphics commands are executed on the separate graphics controller subsystem. The graphics subsystem is preferably connected to multiple remote terminals over a network, and communicates with them without requiring all work to be performed by the central processor which is executing the applications.

The invention as well as a preferred mode of use, and further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a block diagram illustrating a client/server model for a graphical interface;

Figure 2 is a high level block diagram of a central computer system similar to prior art systems which performs graphical display related processing in a central processor,

Figure 3 is a block diagram of a preferred embodiment of a processing system according to the present invention; and

This Sheet Has Been Replaced



Figure 4 is a block data/flow diagram illustrating a software system preferably used in conjunction with the process of Figure 3.

The preferred embodiment of the invention will be described herein in conjunction with a particular implementation which utilizes the X Windows graphical display standard. It will be appreciated by those skilled in the art that the teachings of the invention can be adapted for use with other display systems having similar requirements.

Figure 1 is a block diagram illustrating the client/server model utilized by X Windows. This model includes two main pieces: an X display station 10 and a host system 12. The X display station 10 and the host system 12 are connected through a network 14.

The precise type of network 14 used does not form a part of the present invention. The network 14 can be, for example, an Ethernet connection running TCP/IP. However, networks using other networking protocols, such as Systems Network Architecture (SNA) or Open Systems Connection (OSI), and other communications protocols, such as Token Ring LAN, or Integrated Services Digital Network (ISDN), may also be used.

The X display station 10 includes a network interface 16 connected to the network 14. An X server 18 is connected to the network interface 16 and to a display driver 20. The display driver 20 in turn drives a visual display 22. The display station 10 can be any form of display device which supports X Windows. Examples of stations which are suitable for this task include UNIX workstations, dedicated X terminals, and X emulators running on other types of computers.

The X server 18 is software or firmware which can execute on a full featured work station, or on a dependent terminal, which may be programmable. Descriptions of the performance and functions of the X server 18 are widely available, and the specifics do not form a part of the present invention. In general, the X server 18 interprets requests transmitted over the network 14 through the network interface 16, and causes the display driver 20 to generate output on the display 22. The X server 18 also receives inputs from one or more input devices 24 through the display driver 20.

The functions of the X server 18 are well defined and constant over all types of X display stations 10. The display driver 20 is specific to the hardware of the display station 10, and interfaces directly with the display 22 and the input devices 24. Display 22 typically includes an all points addressable, raster video monitor, and input devices 24 include a keyboard and a pointing device such as a mouse. When the display driver 20 detects an input event, such as a key hit or a mouse movement, the X server 18 may generate a message to be transmitted over the network. Messages may also be generated by the X server 18 in response to some types of requests received from the network.

The host system 12 resides at the other end of the network link. Host system 12 is shown in Figure 1 as being a single system, but it may actually consist of multiple separate systems communicating with the same display station 10 over the network 14. An important function of the X server 18 is to make and maintain network communications links with the one or more host systems 12.

Assuming a single host system, the network attachment hardware is controlled by a network interface 26. The

network interface 26 communicates with Xlib 28 which is a library of relatively low level routines. Xlib 28 is defined in detail as part of the X Windows specification, and may be extended if desired.

A plurality of applications 30 communicate with Xlib 28 through procedure calls. These applications 30 are referred to in the X Windows system as "clients". These clients are the actual applications, such as word processors, computer aided design programs, and spread sheets, which are executed on a computer system. All input and output by the client applications 30 to a human user is preferably performed directly or indirectly through the Xlib 28 procedure calls. Output generated by an application 30 is transmitted by Xlib 28 through the network 14 to the X server 18, with input generated from an input device 24 travelling in the opposite direction. The combination of the Xlib 28 and X server 18 routines handles nearly all of the low level details of the display, with the client applications 30 being freed to perform their primary function without consideration of display details.

In most situations, the applications 30 do not interface directly with Xlib 28, but rather use tool kits (not shown) which provide a high level interface with the applications. The tool kits, in their turn, make calls to the Xlib 28 routines. In providing a high level interface to applications, the tool kits can require a large amount of processor performance. Standard tool kits are made available to applications writers in order to simplify their task, but they appear to the X Windows System as merely another part of the client applications 30.

A considerable amount of processing may be performed within the Xlib 28 routines. Xlib 28 handles drawing of graphics figures, and initiation and maintenance of network communications with the display system 10. A large number

of routines in Xlib 28 are dedicated to the management of windows, including such considerations as display of overlapping windows, exposure events, and window movement across the display. As is well known in the art, these kinds of display processing require significant amounts of processor effort.

Image processing of various types is often performed in X Windows environments. This type of processing is well known to be highly processor intensive. Bit mapped images can be scanned into a file, or an image file can be created by an application. Various types of manipulation, such as image rotation and scaling, can be performed on the images. Image compression and decompression can be done to decrease storage requirements for the image files. This processing is often performed in tool kit routines which are callable by various application programs. It is well known that this type of processing also requires significant amounts of processor time.

Although it is not required, one of the client applications 30 typically is a window manager program.When used, the window manager client handles many details of window positioning and sizing, and movement of windows on the display. The window manager client utilizes many of the window management routines within Xlib 28.

Figure 2 illustrates a computer system such as is typically found in the prior art. The computer system 32 includes a central processor 34 and a network interface card 36. Both are connected to a system bus 38, which also connects other subsystems typically found in a computer system, such as input/output devices, memory, and mass storage (not shown). Network interface card 36 makes physical connection with a communications link 40, which in turn connects to the network 14.

The problems caused by the large processing requirements of Xlib 28 are easily understood with reference to the computer system 32. Central processor 34 must execute all of the Xlib procedures as well as the client applications 30. Whenever communications must be made to or from the network 14, the Network interface card 36 handles the low level details of such transfer. However, every data transfer in either direction must ultimately be handled by the central processor 34.

Since the Xlib procedures are known to be processor intensive, it will be appreciated that the overall system performance of processor 34 will be greatly degraded. The amount of degradation may, in fact, not be a problem in systems which are executing a relatively small number of tasks. However, if system 32 is being used as a central system, with many client applications 30 executing thereon, system performance will be low and system response time will be long. Thus, the use of a processor intensive graphics system such as X Windows is seen to work poorly in an arrangement wherein a single central system supports a large number of simple terminals.

Figure 3 illustrates a preferred computer system which incorporates teachings of the present invention. The system 42 includes a central processor 44 which executes client applications 30 as described above. Central processor 44 is connected to the remainder of the system, including subsystems which are not illustrated, through system bus 46.

A secondary processor 48 is also connected to the system bus 46, and provides a link to the network 14. Secondary processor 48 actually contains a full featured, programmable computer system within itself. The secondary processor 48 contains a bus interface 50 to the system bus 46, and a processor 52. Also included is a network interface 56.

Additional features found in full featured computer systems are also included but not shown, including items such as a read only store, random access memory, control and timing circuitry, and optional mass storage.

The processor 52 is preferably a full featured general purpose microprocessor. For example, secondary processor 48 can be designed around an i960 microprocessor, available from Intel. Additional co-processors (not shown) such as numeric co-processors or digital signal processors, are preferably included within the secondary processor 48 to improve the performance of number intensive graphics functions. The processor 52, alone or in combination with any such co-processors, may very well have more raw computing power than the system central processor 44. This allows the secondary processor 48 to execute a large number of processor intensive software, such as the Xlib library 28, which are required only for driving the X Windows displays. This frees the central processor 44 to execute the client applications 30, a job for which it may be much more highly suited.

System 42 can be implemented, for example. in an AS/400 computer available from IBM. The central processor of an AS/400 is well suited to multitasking of applications, but its efficiency at executing graphics intensive applications is somewhat less. The AS/400 is designed to function as a multiprocessor system, so the use of the secondary processor 48 is a natural extension to the basic system.

Figure 4 illustrates the structural relationships between the various software packages executing on the central processor 44 and the secondary processor 48.Each processor 44, 48 has a bus interface 58 for handling the transfer of data between the central processor 44 and the secondary processor 48. As is known in the art, the bus interfaces 58 may have significant portions implemented in hardware.

The nature of the bus interfaces 58 do not form a part of the present invention, and will be dictated by the hardware used to implement the computer system 42.

The structure diagram of Figure 4 is of a type known in the art, and is read in the usual manner. Each labelled block represents a software package. Data is communicated vertically between adjacent blocks using procedure calls and returns, or by an alternative method.

An application 60 executes on the central processor 44. All X Windows related input and output performed by the application 60 is performed by making the appropriate procedure calls to a tool kit 62, an Xlib stub 64, or an X display data manager 66. Other types of input and output, such as to files stored on mass storage devices, is handled using techniques known in the art.

The Xlib stub 64 takes the place of Xlib 28, and appears to the calling application 60 exactly as Xlib 28. Instead of actually performing any Xlib routines in the Xlib stub 64, the procedure calls are simply passed through to the bus interface 58, for execution in the secondary processor 48. The procedure calls may be reformatted if desired to enhance the efficiency of the data transfer over the system bus 46. When an Xlib call is supposed to generate a returned result, any returned results will be transferred to the Xlib stub 64 over the system bus 46. Any necessary reformatting is performed by the stub 64, and returned to the application 60 in the normal manner. Thus, the intensive processing normally in Xlib is not performed on the central processor 44.

The tool kits 62 may be used in two different ways. A large number of tool kits are presently available in the art, and are used to provide a wide array of standardized and desirable program interfaces. These tool kits 62 im-

plement oft used procedure once, and free the application programmer from having to rewrite code which is common to many applications. The tool kits 62 will, in their turn, call Xlib routines 64, or may communicate information to a tool kit executing on the secondary processor. As shown in Figure 4, the tool kits 62 make Xlib procedure calls by calling the Xlib stub 64, or communicate directly to lower levels by communicating data directly to the bus interface 58.

As mentioned above, the tool kits 62 may be handled in two ways. One is for the tool kit to actually execute on the central processor 44. The other technique is to merely provide a stub to a tool kit executing on the secondary processor, similar in function to the Xlib stub 64. In this case, the actual execution of the tool kit code is performed on the secondary processor 48. The latter approach has the advantage of removing additional processing load from the central processor 44. Having tool kits execute on the secondary processor 48 is especially advantageous for those which perform image processing, due to the processor intensive nature of such work. For various implementation reasons, it may be desirable to have some of the tool kits 62 actually execute on the central processor 44, with others executing on the secondary processor 48 and being represented in the central processor 44 merely by stubs.

The X display data manager 66 is an applications program interface (API) which can be called by the application 60. The X display data manager 66 is similar to a tool kit 62, but preferably provides higher level functions. Instead of responding to single characters, it preferably communicates using larger blocks of data. The data manager 66 preferably allows an application to define display panels. The definition includes input field specifications, output areas, graphics interfaces, and similar features. In re-

sponse to calls from the application 60, the display data manager 66 builds and sends data streams to the secondary processor 48 which define entry fiels on the display panels, or requests screen or input field data. In return, the display data manager 66 receives data streams from the secondary processor 44 and processes them for return to the application 60. These returning data streams include such functions as returning input field data and processing special keyboard functions such as Help.

The concept of data streams transmitting large quantities of information is known in the art, and an example of a display data manager which functions in a similar manner to X display data manager 66 is described in **Application System/400 Programming: Data Description Specifications Reference (DDS)** available from IBM. The display data stream used in the AS/400 DDS display data manager is the 5250 data stream, described in IBM publication IBM 5250 Information Display System Functions Reference Manual.

Instead of, or in addition to, the use of data streams to communicate information with the secondary processor 48, the X display data manager 66 can make standard Xlib procedure calls to the Xlib stub 64. These procedure calls are handled in an identical manner to those which are made to Xlib by the tool kits 62, or directly by the application 60.

The software applications which are executed in the secondary processor 48 are shown in the lower portion of Figure 4. The Xlib library 68 actually executes on the secondary processor 48, and communicates with the network interface 70. The Xlib routine 68 may be called directly by the Xlib stub 64 through the bus interfaces 58, or from various applications executing on the secondary processor 48.

As is known in the art, certain X commands pass through Xlib, directly between an application and the X server 18. Commands of this type, which are specially signalled by the applications and the X pointing device until an event such as a mouse button click occurs which requires the attention of the secondary data manager 74 or the application 60.

As described above, some tool kits 78 may be implemented so as to actually execute on the secondary processor 48 if desired. These tool kits 78 may be invoked by stubs running on the central processor 44, or may be invoked directly by the secondary data manager 74 or any other application running directly on the secondary processor 48. Tool kits 78 which perform image processing, such as image decompression, scaling, and image manipulation, are well suited for executing on the secondary processor 48 because of their heavy demands on the processor.

As described above, the window manager is a special client application which can be used to control the windows of multiple other client applications. Execution of the window manager is processor intensive. The window manager client 80 may be executed on the secondary processor 48, relieving the central processor 44 of all of the burden of executing the window manager. The window manager client 80 is a registered window manager within the X Windows system, and makes procedure calls to Xlib 68 to communicate with the X server 18. Since execution of the window manager client 80 is processor intensive, offloading it entirely to the secondary processor 48 greatly relieves the burden on the central processor 44.

As will be appreciated by those skilled in the art, the functioning of the various software packages is not greatly changed from that of prior art systems. Xlib is the same library, it merely executes on a different proces-

sor. The use of stubs, such as Xlib stub 64, is generally known to those skilled in the art, although not in a context such as the present one. The remainder of the software system, who's structure is shown in Figure 4, can be easily implemented by those skilled in the art by adapting well known, widely available software to the structure described. This means that the amount of code which must be rewritten to implement the described system is moderate relative to the large performance gains which can be achieved.

Since the significant burden of the details of the X Windows systems are offloading from the central processor 44 to the secondary processor 48, the central processor 44 is capable of supporting a large number of X Windows applications. This allows a system such as computer system 42 to be used centrally to support a large number of X display stations which can be connected over a Local Area Network. This can provide cost savings over the use of a larger number of high cost work stations. In addition, if a high capacity central machine, such as an AS/400, is already available, X applications can be run on X display stations for a low incremental cost by connecting them to an existing network.

The embodiment described above assumes that the central processor and the secondary processor reside in a single computer system and communicate over a system bus. An alternative system arrangement which utilizes the concepts of the present invention provides for the secondary processor to be located in a separate machine from the central processor, possibly at a physically remote location. It will be appreciated by those skilled in the art that the described functional split between the two processors results in a relatively minimal amount of data transferred over the system bus. This result can be used to provide a remotely located secondary processor connect-

ed to the central processor by a high speed serial communications link which performs in almost an identical manner to the system described above. In effect, the central and secondary processors operate concurrently with message passing linking their operation, and it does not matter whether the messages are passed over a system bus or a serial link.

This allows, for example, design of a system in which applications are executed on a central mainframe computer an the X display terminals used for input/output are located in a different city. A machine which is connected to the display terminals over a local area network can contain the secondary processor. The secondary processor and mainframe can communicate over a serial link. Since the data transferred between the central processor and the secondary processor is usually significantly less than that transferred between the display terminals an the secondary processor, overall system performance will not suffer too much. This allows a system executing X applications to drive terminals located in many different remote locations, which cannot be done using available prior art systems.

* C L M S *

CLAIMS

1. A computer system, comprising:

a central processor for executing application programs;

a plurality of graphics terminals; and

a display controller processor in communication with said terminals and said central processor, wherein output generated by the executing application programs is sent from said central processor to said display processor, which in turn performs graphics processing thereon and sends the processed output to one of said terminals, and wherein input generated at a terminal is sent to said display processor, which in turn performs any necessary graphics processing thereon and sends processed input to the executing application program.

2. The system of Claim 1, wherein said terminals are connected to said display processor by a communication network.

3. The system of Claim 2, wherein said display controller processor comprises :

a network interface connected to the communication network;

a library of executable procedures for performing graphic functions for communication to said terminals through said network interface; and

This Sheet Has Been Replaced

* C L M S *

a communications interface for transferring information between said executable procedures and said central processor.

4. The system of Claim 3, wherein said executable procedures are invoked by application programs executing on said central processor, wherein an invocation is made by transferring a message from an executing application through said communications interface, and wherein results are returned to the executing application through said communications interface.

5. The system of Claim 3, further including at least one application program executing on said display controller processor.

6. The system of Claim 5, wherein the application program executing on said display controller processor comprises an image processing tool kit.

7. The system of Claim 3, wherein said terminals are responsive to X Windows System commands, and wherein said library comprises XLIB.

8. The system of Claim 2, wherein the communication network comprises a local area network.

9. The system of Claim 1, wherein said central processor and said display controller processor communicate through a computer system bus.

10. The system of Claim 1, wherein said central processor and said display controller processor communicate over a serial communications link.

11. The system of Claim 1, wherein at least one input event is completely processed in said display con-

This Sheet Has Been Replaced

* C L M S *

troller processor without intervention from said central processor.

12. A method for performing input/output between an application executing on a computer system and a graphics display terminal, comprising the steps of:

executing the application on a central processor;

when output is generated by the application, communicating such output to a separate display controller processor;

on said display controller processor executing graphics display routines for translating generated output into a form suitable for driving a graphics terminal;

communicating translated output to a graphics terminal for display;

when input is generated at the terminal, communicating the generated input to the display controller processor;

on the display controller processor, translating generated input to a form suitable form communication to the application; and

communicating the translated input to the central processor for use by the application.

13. The method of Claim 12, wherein communication between the central processor and the display controller processor is performed over a computer system bus.

This Sheet Has Been Replaced

14. The method of Claim 12,wherein communication between the central processor and the display controller processor is performed over a serial communications link.

15. The method of Claim 12, wherein communication between the display controller processor and the terminal is performed over a local area network.

16. The method of Claim 15, wherein the terminal is responsive to X Window System commands.

17. The method of Claim 16, wherein both of said translating steps comprise executing Xlib routines on the display controller processor.

18. The method of Claim 17, further comprising the step of:

executing, on the display controller processor, additional procedures which in turn invoke Xlib procedures.

19. The method of Claim 18, wherein such additional procedures comprise image processing routines.

20. The method of Claim 17, further comprising the step of:

on the display manager, executing an X Windows System Window manager client application.

21. The method of Claim 12, wherein at least one input event is completely processed on the display controller processor without transferring any information to the central processor.

This Sheet Has Been Replaced