A DISTRIBUTED SCIENTIFIC VISUALIZATION PARADIGM FOR HETEROGENEOUS COMPUTER NETWORKS

A Thesis

by

MARK WAYNE LENOX

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 1990

Major Subject: Electrical Engineering

A DISTRIBUTED SCIENTIFIC VISUALIZATION PARADIGM FOR HETEROGENEOUS COMPUTER NETWORKS

A Thesis

by

MARK WAYNE LENOX

Approved as to style and content by:

[illegible]
Bahram Nassersharif
(Co-Chair)

[illegible]
Karan Watson
(Co-Chair)

[illegible]
M. Styblinsky
(Member)

[illegible]
P. Cantrell
(Member)

[illegible]
J. Howze
(Head of Department)

December 1990

ABSTRACT

A Distributed Scientific Visualization Paradigm for Heterogeneous Computer Networks.

(December 1990)

Mark Wayne Lenox, B.S.E., Arizona State University

Co-Chairs of Advisory Committee: Dr. Bahram Nassersharif
Dr. Karan Watson

Current methods of supercomputer based scientific visualization place unnecessary load on the supercomputer. By separating the computational component from the graphic rendering component, and distributing both processes over a network, overall scientific visualization performance increases. Furthermore, the supercomputer CPU load and memory requirements are decreased, allowing the solution of larger computational problems. A distributed scientific visualization paradigm is developed for a heterogeneous computer network, and the performance characteristics of this model are compared against the X Window System.

DEDICATION

To Katey

ACKNOWLEDGEMENT

The work presented in this thesis could not have been accomplished without help from the staff of the Texas A&M Supercomputer Center. Thanks are in order to the director, Dr. Bahram Nassersharif, as well as Mike Bolton, Spiros Vellas, Gynger Ingram and Victor Hazlewood for their help and patience.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

Complex scientific calculations are sometimes more easily visualized graphically than numerically. Unfortunately, the computers that are most capable of performing advanced calculations (supercomputers) are least capable of rendering graphical output.

Software systems exist by which supercomputers can do calculations, and render graphical output. Software like the X Window System allows applications programs the flexibility of displaying data on any X display server on a network [25]. Other graphics packages, like NCAR graphics and DI-3000 for instance, allow applications to display output on graphics terminals [8,6].

These software systems have inherent drawbacks, however, that make their use non-optimal for extremely high performance computing. In systems like X and NCAR, the graphics rendering is actually done by the supercomputer. Since graphics rendering code is somewhat convoluted (and large), it does not vectorize well, and serious performance penalties result both for the rendering code, and the overall application [14,9]. The availability of dedicated graphics hardware to perform rendering operations would remove the performance penalties, but is currently unavailable to most supercomputers.

In this work, I propose to develop a distributed paradigm for scientific visualization. Supercomputers do well on ordered sets of data with minimum I/O. Graphics workstations have specialized hardware to render graphics. A distributed scientific visualization paradigm allows both machines to perform at their best. The supercomputer runs a highly vectorized calculation kernel that dumps binary data over a network to a graphics workstation that processes the information, and renders the graphics in near real-time. The end result is that the supercomputer can run to its maximum capacity, and the graphics workstation can utilize on-board graphics hardware to accelerate the rendering process.

This thesis follows the format of ACM Transactions on Graphics.

The system as a whole is referred to as SP (short for Scientific visualization Paradigm). The software running on the workstation is referred to as SPD (SP Daemon), and the supercomputer resident communication libraries are know as Libsp.

Other Projects

There have been other projects associated with distributed scientific processing. The most notorious of these projects includes the X Window System from MIT, apE from OHSC, Chemtool from Cray Research, Inc., and NetCDF from UCAR [25,9,5,24].

NetCDF

NetCDF is a system created by the University Corporation for Atmospheric Research (UCAR) for the network transparent movement of scientific data between computers on a heterogeneous network [24]. Although NetCDF does not actually allow graphical visualization, it pioneered the concept of machine independent network transparent communication of scientific information that is necessary for a distributed scientific visualization paradigm to operate.

NetCDF uses a system developed by Sun Microsystems called external data representation, or XDR, to accomplish the data transfers. This system uses IEEE floating point formats for all data representations, and assumes that only 8-bit bytes can be interpreted in a consistent fashion.

CRI Chemtool

An outstanding example of the concept of distributed visualization in use is the Chemtool software developed by Cray Research, Inc [5]. Chemtool allows molecules to be graphically built on a Silicon Graphics IRIS workstation, then submitted to a Cray for analysis. During the analysis phase, certain parameters (like temperature) can be changed, and their effects seen on the graphic molecule (disassociation, or oscillation for example). Unfortunately, Chemtool is a specialized application for computational chemistry. A general purpose paradigm is needed to provide the same functionality to a larger number of scientific visualization problems.

MIT X

In the mid 1980s, researchers at MIT started working on a network-based windowing system [25]. The result is the X Window System, a distributed graphics paradigm that allows data to be generated on a displayless mainframe, transmitted over a network, and displayed on a graphics workstation. The primary problem with X, however, is not functionality, but equity in processor loading. Figure 1.1 shows the obvious disparity in processor loads between the supercomputer and the UNIX workstation.

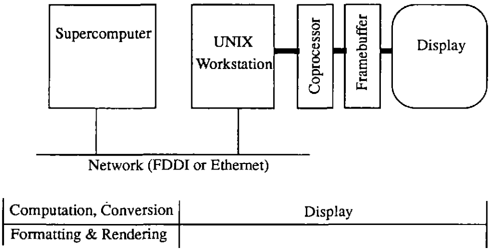

Fig. 1.1. X Window System communication and processing path

Although X provides the necessary network transparent functionality, it has several drawbacks that make it unsuitable for scientific visualization: First, X based programs are complex. Second, the convoluted nature of interactive software is not easily vectorizable, and therefore does not allow a vector supercomputer to compute at it's full potential speed.

Third, since every graphics primitive translates to a network packet, complex displays are slow to draw, especially over heavily loaded network connections. Finally, graphics rendering must be done on the pixel level by the supercomputer, wasting cpu cycles and memory that could be used to do more useful work [22].

Sun VX/MVX

Sun Microsystems has implemented a hardware product called VX/MVX that uses multiple processors to perform visualization tasks [33]. It is not a heterogeneous distributed environment, however, in that all the processors are located within a single Sun workstation, as shown below in figure 1.2.

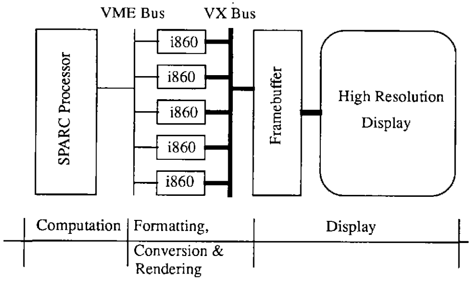

Fig. 1.2. Sun's VX/MVX design

The VX/MVX paradigm exhibits some of the characteristics needed in a distributed visualization paradigm, including equity in processor loading. The separate i860 processors can be individually programmed to render data supplied by the SPARC processor over the VME bus [13]. It is unsuitable for supercomputer based visualization due to the lack of a direct VME connection between the VX/MVX subsystem and a supercomputer, but the idea is generally correct.

Apollo Virtual Graphics Pipeline

In the SIGGRAPH proceedings of 1988, Douglas Voorhies of Apollo proposed a virtual graphics paradigm to increase graphics rendering throughput in desktop workstations [32]. It is similar to the Sun proposal, but concentrates more on increasing overall throughput with more equitable processor loading and a flexible interface rather than just increasing the number of processors. This paradigm is shown in figure 1.3.

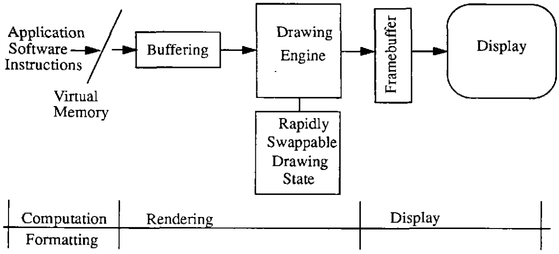

Fig. 1.3. Virtual graphics paradigm proposed by Voorhies

The hardware solution proposed by Voorhies gives the necessary functionality and removes some of the drawbacks of conventional systems through the use of the virtual graphics paradigm. In a system using virtual graphics, all processes are fooled into believing that they have a dedicated graphics coprocessor. This provides for flexibility, combined with increased throughput and singularity of processor tasking. The whole system works better because the individual components are used exclusively for what they were designed to do. Unfortunately, Voorhies' solution is as unworkable in a supercomputing environment as the

Sun VX/MVX because of the prohibitive costs of any hardware implementation.

OHSC apE

The Ohio Supercomputer Center Graphics Project released apE 2.0 in September of 1990. This project followed design criteria similar to those established for SP. Abstraction of scientific data, distributed processing pipeline, and minimization of supercomputer resource demand are all design features of apE [17].

apE, which stands for animation production Environment, abstracts data in the form of the FLUX language [17]. FLUX has built in capabilities that make it useful for both scalar and vector field representation. The apE system, however, is almost strictly limited to field representations, and contains no provisions for other potential visualization models. Limitations in the FLUX communication language constrict apE in this regard.

The implementation of apE involves a distributed processing pipeline that includes a supercomputer performing strictly numerical computation, and a workstation to perform post processing and graphics rendering [9]. This feature allows the supercomputer to perform at peak speeds by minimizing extraneous overhead. Actual scientific studies done using apE have shown that considerable levels of performance can be achieved through the distributed paradigm that it uses [14].

ApE was a valid attempt to create a general purpose distributed scientific visualization paradigm, however, it failed for several reasons. The first reason is that it was designed for the visualization of field data. The FLUX language is explicit on this point. Second and foremost, the current performance status of supercomputers and UNIX workstations requires that software to do distributed scientific visualization be as optimized as possible for the particular necessary operations. The apE system with its very flexible FLUX language is too verbose, and requires a significant amount of overhead to operate. The desired architecture requires that only scientific data be sent over the network, not just another representation of the picture (which is what FLUX does). Ideally, the workstation provides enough post processing to completely visualize the data.

Design Methodology of SP

The solution to this problem, can be found through the network transparency features of netCDF, and an emulation of the hardware virtual graphics paradigm proposed by Voorhies, and the VX/MVX architecture implemented by Sun Microsystems. The virtual graphics paradigm can be duplicated in a networked supercomputing environment, substituting a fast network connection for the virtual memory interface, as shown in figure 1.4. In this environment, a UNIX workstation can be programmed to support virtual data connections to any number of networked supercomputers. As a final step, the workstation can be programmed to not only render data, but to do final analysis and storage of that data, as proposed by the VX/MVX paradigm. This feature off-loads computational requirements from the supercomputer, allowing more efficient allocation of expensive CPU and memory resources.

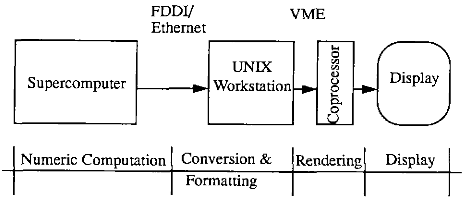

Fig. 1.4. Distributed visualization paradigm

When implemented in this form, a distributed visualization paradigm provides the following functionality: A user supplied program running on the supercomputer generates numeric data. Libraries located on the supercomputer format the data, and transport it over the network to the UNIX workstation for further processing. The workstation processes the information, and DMAs rendering commands to the graphics coprocessor, which displays the results.

There are many benefits to a distributed visualization paradigm in this form. First, software requirements on the supercomputer side are very small, releasing CPU and memory resources that can be better allocated solving problems. This feature also has the benefit of increasing the portability of the entire project. Second, since only the binary results of computations are transmitted, network traffic is minimized. Third, all rendering commands occur across DMA channels for higher performance. Finally, near real-time graphics processing is supported through the systems ability to continuously process, render, and store data.

The desirable nature of this paradigm is not in the addition of functionality, but the equity of processor loading. As modern scientific workstations become more powerful, they can share the processing load. Previous systems such as X, NCAR, and DI-3000 place nearly all of the processing load on the supercomputer [25,6,8]. A distributed visualization paradigm, however, places loads across all the processors in the system [11,23], and has the added benefit of load balancing. The jobs that each processor executes are explicitly tailored to the architecture of the processor. The vector supercomputer runs highly vectorized floating point computational software. The UNIX workstation runs general purpose conversion and formatting routines, and the graphics processor renders data to the display.

SOFTWARE DESIGN

Introduction

The distributed scientific visualization paradigm (SP) was designed using structured design techniques and implemented with the object oriented techniques described by the Xt Intrinsics [1]. The system as a whole consists of two parts, the communication libraries that exist on the supercomputer (Libsp), and the daemon process that runs on the workstation (SPD).

This particular implementation of SP is concerned with Cray to Sun connections. Therefore, SPD is implemented on a Sun workstation under SunOS, and Libsp is implemented on a Cray Y-MP under the UNICOS operating system. The methods presented here are not tied specifically to these platforms, however, in their implementation. All software was written in the C programming language [4] and all graphics calls are to the X Window System [19-21]. The networking portions of SP were written to utilize the industry standard Berkeley Socket libraries [16,27], so in this respect, SP is not tied to a specific hardware platform. In fact, Libsp was originally implemented on a Sun workstation before final implementation on a Cray Y-MP, demonstrating the inherent portability.

SP Overall Organization

The primary purpose of SP is to move raw data from a scientific application on the Cray to a Sun workstation to be viewed. On the Cray, application software formats data, and sends it across the network with Libsp. See the appendix on Libsp for more details. The overall process is shown in figure 2.1.

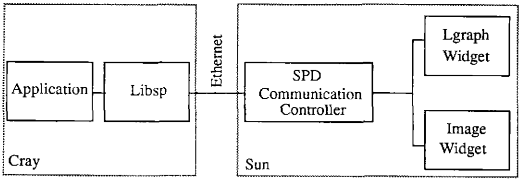

Fig. 2.1. Data flow diagram

On the Sun, data is received by an SP daemon (SPD) from an ethernet socket connection, formatted, and sent to the appropriate display widget for physical rendering.

SPD Organization

The SPD is organized according to the constraints provided by the X Window System Xt Intrinsics libraries [20], and is shown in figure 2.2. It provides functionality in three basic areas: communications, visualization, and general data manipulation utilities. The communications routines consist of a connection handler, and a stream communication handler. The display routines consist of a series of visualization widgets to perform physical rendering, and the general utilities provide data storage and export capabilities.

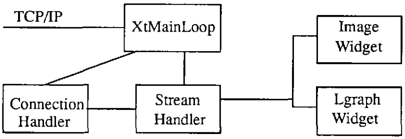

Fig. 2.2. SPD organizational diagram

SPD communications support consists of routines that open and maintain TCP/IP sockets for network communication [27], and a device driver /dev/graphics for local operation. Socket operations require two virtual data streams: a connection request stream, and a data stream, therefore, there are two communication handlers. When a remote machine makes a connection request to the SPD, the connection handler processes that request, creating a communications context, and initializes a data stream associated with that request and the remote machine. All further communication with the remote machine is done with the stream handler.

The stream handler reads information from the remote machine, and converts it to a form that can be used by the visualization widgets.

Libsp Organization

Libsp is a collection of subroutines that reside on the Cray that give programmers easy access to the SP protocols. Libsp divides functionality across two example visualization areas: Lgraphs and Images. Lgraphs are X-Y line graphs, and Images are color collections of raster data. Very little front-end processing is done by libsp. It serves almost strictly as a communications library.

All software written to utilize Libsp first must initialize an SP protocol channel to an SP visualization server. This is done with the SpInitialize command. This command parses the command line parameters, and removes parameters that are specifically directed at SP operations. Next, SpInitialize opens a socket channel [27], and binds it to the address of an SPD visualization server. If all is well, SpInitialize returns a handle to the visualization server called Display. When all operations are complete, an SpShutdown command is issued. This command notifies the SPD visualization server that all computation is complete, and that all communication channels should be closed. When SpShutdown returns, the program can be terminated without potential loss of data.

Once a communication channel is open, the visualization models can be invoked with either a call to SpLgraphOpen, or SpImageOpen. Data can be written to the visualization models, and finally, they can be closed with calls to SpLgraphClose, or SpImageClose.

Data flows through libsp in the manner shown in figure 2.3. A communication channel is opened and bound by SpInitialize. An SpXXXXXOpen command instantiates a visualization model. Data is written to the model with SpXXXXXAddData command, and the channel is closed with SpShutdown.

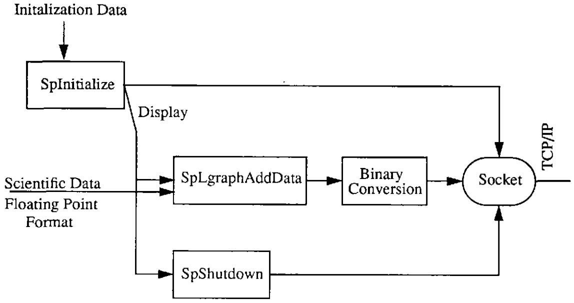

Fig. 2.3. Libsp data flow diagram

Writing Distributed Visualization Applications

Writing a distributed visualization application that utilizes the SP methodologies is a four step process:

Define the precise form that the visualized data should take, and implement an X11R4 widget to render that data [1].

Determine the scientific data that is necessary to generate the graphics, and add the appropriate packet types to the SP communications protocols [27].

Add any necessary subroutines to Libsp for formatting and setup of a display model instance [16].

Implement the computation kernal for the supercomputer.

These four steps obviously involve considerable detail. The implementation of a distributed visualization display model is a non-trivial task, however, the ability to use a standard X11R4 widget to render data provides an easy development solution that can be ported to a large number of different systems.

Networking

SP functions as a distributed system. The libsp communications library and the SPD work together to get their job done. This dependence puts a large amount of reliance on the associated network connection. If networking fails, so does the SP system as a whole. For this reason, TCP/IP was selected as the communication protocol. For sockets opened in TCP mode, correct data transmission is guaranteed [27]. If the channel fails, it will close completely. Errors are detected, and corrected at the protocol layer, so no ECC code is necessary at the application layer. Other protocols such as UDP, or IP without TCP do not have this feature, and are therefore unsuitable for this application without significant application overhead.

BENCHMARK

In order to measure the performance of SP, it was benchmarked against the only other currently available distributed graphics paradigm, the X Window System. The characteristics measured include: Cray user CPU time, Cray system CPU time, Cray memory, network loading, and some qualitative comments on application responsiveness.

Examples

Two visualization models were developed for SP for testing purposes, a line graph model (Lgraph), and a scalar field model (Image). Both models were written as Xt widgets for inclusion into SP, and also for ease of porting to the Cray environment for direct testing without the SP communication interface. More precise definitions for these widgets can be seen in the section entitled Visualization Widgets.

The Lgraph model was chosen because it represents a visualization method with a high level of abstraction. The programmer has only cursory control of the way the graph looks. The data set that the graph represents is made up of a series of floating point values. The precise method of rendering the graph is unimportant to the application programmer. Another feature of the Lgraph model is that it utilizes loosely coupled asynchronous communications to improve user responsiveness.

The Image model was chosen because it is nearly the antithesis of an Lgraph model. Image models by definition include very low levels of data abstraction. The application programmer knows that the data in an array will directly translate to a color matrix on the screen. The transport of that matrix is unimportant, but the size and access methods are well known. The Image model uses a tighly coupled communication scheme to improve data transfer performance and reduce overhead.

Methods

To benchmark SP against X, six example programs were developed. Three examples were written to use the Lgraph model, and three were written to use the Image model. Of the examples in each class, one used the SP communication interface, one used the X interface directly, and one provided no display output at all.

The Lgraph example renderes a sin and cos function between the value 0 and a variable L. For each integer increment of L, one hundred data points were calculated and rendered to the display. Increasing L linearly increases the computational and rendering requirements linearly. An example of the Lgraph demo is shown in figure 3.1.

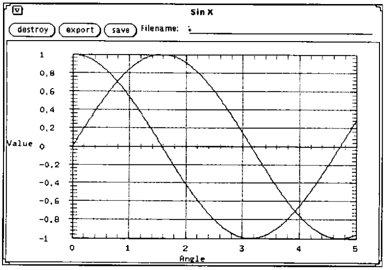

Fig. 3.1. Line graph example

The Image example rendered a sequence of frames depicting the diffusion of gaseous particles across a space. The image size is the variable in this case, however, increasing the number of pixels in the image linearly will increase the total computational and rendering requirements in a non-linear fashion. As the image gets larger, more frames are computed to show the motion with the same degree of smoothness. An example image is shown in figure 3.2.

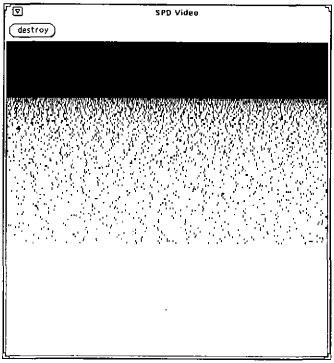

Fig 3.2. Image example

The performance characteristics of all the examples were measured at varying levels of activity. The Lgraph models were tested rendering output of data sets sized between 5000 and 15000 data points. The Image models were tested rendering images between 10,000 and 250,000 pixels.

Performance characteristic measurements were taken on the Cray using the UNICOS utilities timex, procstat, and procrpt. Timex performs CPU utilization monitoring for both user and system CPU time [30]. Procstat performs memory utilization monitoring, and procrpt generates a report from the data generated by procstat [31].

Network loading was measured using the SunOS etherfind command on a Sun workstation [28]. Etherfind reports on each data packet that is transmitted between a given set of machines. Information is provided about port connection information, the number of actual bytes transmitted, and the protocol used. Further filtering of the data generated by etherfind is used to strip the data of all traffic not associated with a particular test, and to sum the number of bytes used in the complete run.

RESULTS

SP met all functional requirements by decreasing Cray CPU demand, memory utilization, and network loading.

In order to insure that SP was working correctly, a number of small test programs were developed to exercise the individual operational units of SP. Six of those test programs became the benchmark suite, and were subject to critical evaluation, but the others yielded some of their own data as well. One of the most important finds was the quantification of system CPU requirements involved in network communication processing overhead. There is a direct relationship between the amount of data written directly to a network socket and the system CPU usage. Overhead is expected in all I/O operations, but network overhead is higher than most. Through experimentation, it was found that between 0.5 and 5 microseconds of CRAY system CPU time were required per byte of data moved across the network. These values are highly dependant on buffer size, and the priority given to the I/O processes on both machines (coupling). Larger buffers augmented with tight coupling performed the best, and smaller loosely coupled buffers performed the worst. Reducing the overhead involved in the data transfers only becomes possible, however, if two conditions are met. The process in the receiving workstation must be programmed to accept and process data in large blocks with high priority (tight coupling), and the writes must be very large (greater than approximately 10KBytes). If either of these conditions are not met, the Cray CPU overhead will gravitate toward the 5 microsecond/byte figure. This agrees well with the data collected on the benchmark suite.

Unfortunately, when dealing with scientific visualization problems, large amounts of data are usually involved, therefore the network overhead both in terms of actual physical network loading as well as the CPU load associated with the transfer of data can become costly if not limiting. Writers of SP display models should keep this information under consideration.

Overall Comparison

Although there is a large difference between the capabilities of X and SP, great care was taken to make a performance comparison meaningful. The X and SP versions of the examples perform exactly the same computations, and render exactly the same output sequences. It does not matter how much flexibility one system has over the other, provided the same functionality exists to solve the problem on both platforms. The additional overhead necessary for one system over the other is simply wasted on the problem if it is not necessary for the solution of a given problem.

Image Performance

The Image model was able to take advantage of the SP protocols to severely reduce the load on the Cray and the attached Ethernet network. Since the Image test represents a non-linear function, it is expected that the results should be non-linear as well.

Figure 4.1 shows the CPU time demands to render the test sequence of images. At the high end, CPU time requirements are dropped by an order of magnitude. Most of the gains in this area were caused by the simplification that SP provides. Furthermore, the SP image processing routines that are present on the Cray side are vectorized, while X has little or no vectorization. The obvious non-linearity of X processing time comes as no surprise. The Image test case is a non-linear function, although it can barely be detected in terms of CPU performance. The processing requirements of X increase linearly with the size of the image to be processed, however, when the processing requirements of X are multiplied by the non-linear processing requirements, the resulting overall graph is obviously non-linear.

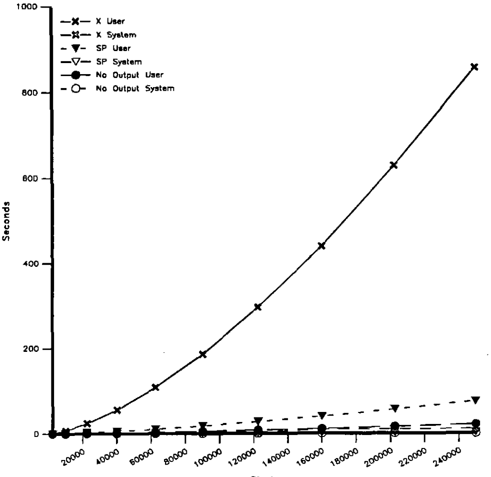

Fig. 4.1. Cray CPU requirements for image processing

Figure 4.2 shows Cray memory requirements for the test sequence to be approximately twice as high for X as compared to SP. Both X and SP increase memory demands linearly as the image size increases. The non-linearities of the Image test sequence do not have any effect here, so the graphs do not increase exponentially. The slight jog in the memory allocation curve for both SP and X indicates a UNICOS problem (or feature) in memory allocation, and may be a topic for further study.

In figure 4.3, network traffic is shown to be significantly lower using SP. Again, the non-linearities of the Image test case make themselves apparent. Curiously enough, X does not move as much data as the theoretical maximum would permit. Some optimization does in fact occur. SP, however, using the delta compression algorithms, can potentially do much better, as it does in this example. For more information about delta compression, see Appendix D. In the worst case, where every pixel on the display changes every frame, SP requires the theoretical maximum data transfer. The actual transfer requirements in the theoretical worst case is the number of pixels in the frame times the number of bits per pixel divided by 8 bits per byte. The maximum rate is limited to the speed of the physical connection. The SP Image example does not allow for cases with less than eight bits per byte, and will transfer data in an 8 bit format in that case. In the theoretical worst case, X can in fact provide slightly less network traffic to render a particular sequence. Note, however, that once SP transfers a frame, no further communication is required if the frame needs to be redisplayed. If an X window becomes suddenly exposed, the entire frame must be retransmitted. Therefore, in a real-use environment, the network traffic requirements of X can multiply themselves several fold over SP.

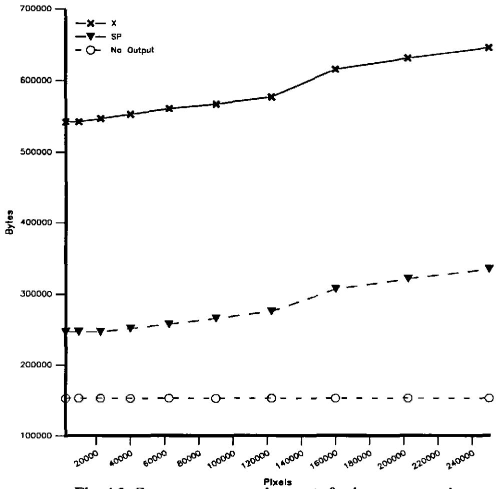

Fig. 4.2. Cray memory requirements for image processing

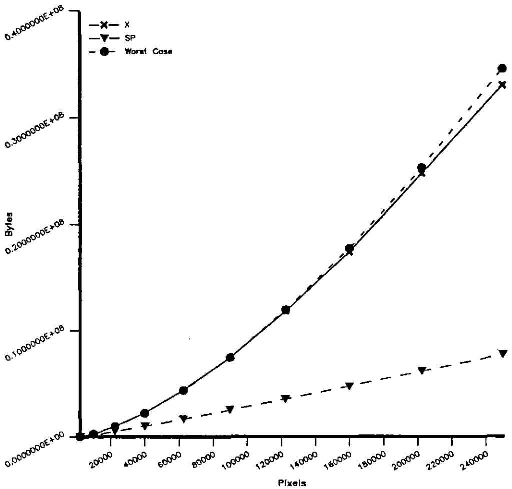

Fig. 4.3. Network loading for image processing

Lgraph Performance

The Lgraph model took advantage of the SP protocols in a different way than the Image model did. Due to the high level of abstraction that the Lgraph model provides, initial communication requirements can be slightly higher, but memory and CPU requirements can be dramatically decreased in a real-use environment. The Lgraph model requires proportionally more system resources as the data set size increases.

Figure 4.4 shows Cray CPU requirements to render the graph test sequences under a moderate, sustainable load (500 data points/sec). The CPU savings associated with running SP for a single case are significant, amounting to approximately 6 Cray CPU seconds in the 15,000 data point test case over X. In fact, SP requires only a fraction of a second of user time, more than the no output case. An additional two seconds of system time was accumulated due to the addition of I/O. Furthermore, once complete, the SP computations never need to be performed again. Slight variations in the system time are caused by network uncertainty.

A test of the limits of capacity in the SP Lgraph display model shows what happens when the visualization pipeline is heavily overloaded. Figure 4.5 shows Cray CPU requirements to render the test sequence of graphs in a high load (40,000 data points/sec) environment. Two points of interest on this graph make themselves immediately apparent: abnormally high system CPU requirements for SP, and highly variable system CPU requirements for X.

First, the abnormally high system CPU requirements of SP were not expected. It has been proposed that the high system time condition is caused by the relative absence of data buffering, and therefore an abnormally high number of I/O requests. In order to minimize display lag, data is only buffered on a point by point basis. To test this proposal, buffering was increased to 400 data points per write (10 KByte buffer). These changes reduced the number of I/O requests by factors up to 400, but had no real effect on the system CPU requirements. The values did vary by +/-.01 seconds, but this represents a change of less than one percent, and is considered negligible. The actual cause of the high system CPU time is the high I/O wait times associated with an overloaded connection. While in an I/O wait condition, UNICOS continuously polls until the write request is granted, driving up the system CPU demand. Increasing the CPU load above the trivial case (and therefore reducing the number of data points/sec to be rendered) reduced the system CPU time reduced to the normal values in Figure 4.4.

The variability of the system CPU requirements for X are expected. It is caused by the dynamic nature of X. Since neither the attached network, nor the Cray itself are dedicated systems, differing loads could interfere with the timing and dynamics that X depends on for operation. In this particular case, the Cray process requests the X server to open a window, and the X server immediately responds with a window ID. At some random time in the future, the X server sends an expose event to the Cray process to tell it to draw on the window. During the random time between the window open and expose event, the Cray is continuously processing and generating numbers. When the expose event occurs, the Cray sends X commands to render the graph in the specified window. From that point on, individual commands are sent to modify the window to fit new calculated information. The wide swings in system CPU time are caused by the variability in both the volume and the consistency of network traffic, which previous results have shown to be directly related. A graph showing system CPU time for three runs of the X example is shown in Figure 4.6.

Specific research into both of these problem areas could provide interesting information about the nature of UNICOS networking. A suggested rule of thumb to avoid the performance penalties associated with overloading the communications channel is to keep the data flow to less than 1000 data points (one X and two Y values) per second of Cray CPU time.

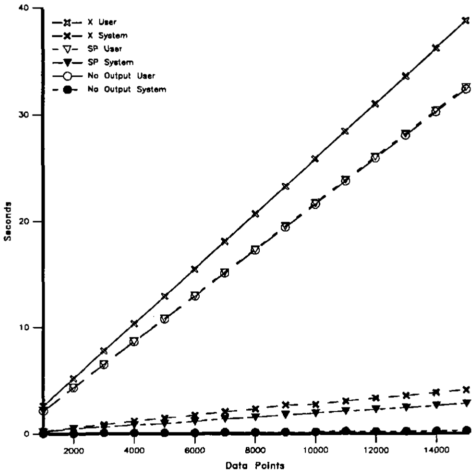

Fig. 4.4. Cray CPU requirements for line graph processing

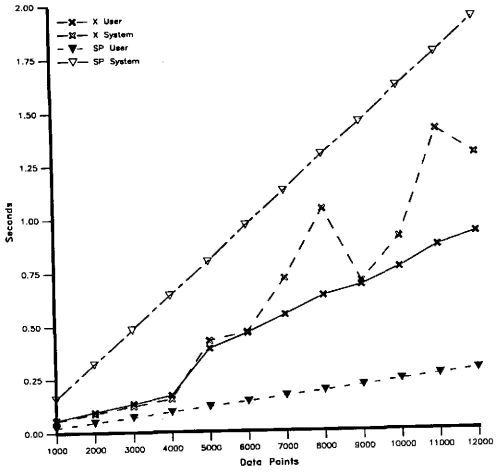

Fig. 4.5. Cray CPU requirements for line graph processing in an overloaded system

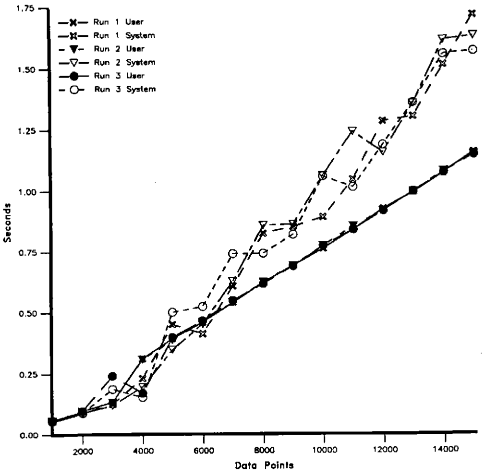

Fig. 4.6. Cray X CPU requirements in an overloaded system

Figure 4.7 shows memory requirements for X versus SP. Since the Lgraph model does not require additional memory to perform calculations, but simply transfers the data that it is told to, memory requirements do not increase with data set size as X does. In this case, a linear increase in the data set size increases the X memory requirements in a linear, but slightly variable fashion. The variability is caused by the X responses to changes in a dynamic system.

Figure 4.8 shows the network loading generated for the Lgraph test sequences. Oddly enough, X actually requires less network traffic for a single rendering of an Lgraph than does SP. This is caused by a fundamental difference in philosophy. The SP protocols transmit all the information required to precisely render the graph in any form. The SP daemon receives and saves all of that information locally so that the graph can be easily regenerated without further communication. This improves response time from a user standpoint. X transmits only enough information for a particular rendering of the data. That information is not saved, so if a redraw is required, all the data must be transmitted again. With SP, all network traffic is up-front, but with X it continues until the application is completely closed. In a real operating environment, these peripheral transfers can cost many times what the up front costs are.

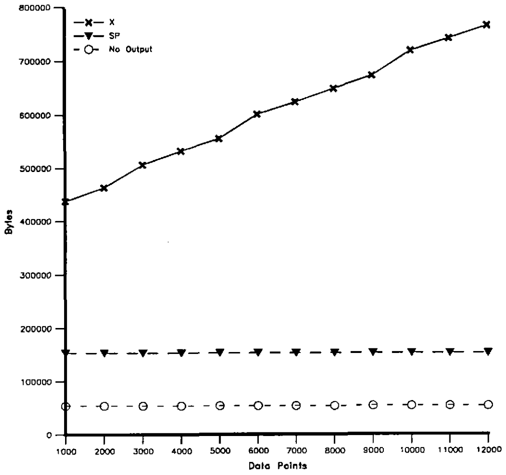

Fig. 4.7. Cray memory requirements for line graph processing

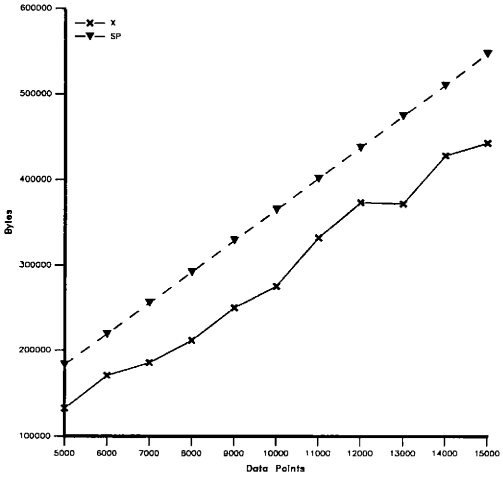

Fig. 4.8. Network loading for line graph processing

CONCLUSION

SP and X represent two distinct philosophies to solve the same problem: distributed scientific visualization in near real-time [15]. SP operates with a very high level of data abstraction, while X performs the same operations with little or no data abstraction. Once an SP job has completed, the connected workstation contains all the necessary information to continue interactive visualization and manipulation without supercomputer intervention. X, however, requires that the supercomputer continue to process until all computations and user manipulations of the data are complete. These additional requirements that X imposes add unnecessarily to the overall cost of performing scientific calculations on a supercomputer in terms of money and accuracy. SP removes these problems and allows the supercomputer to perform more efficiently.

Data Buffering

When and how data is buffered when it is generated from the supercomputer is of great significance to the performance of SP. The larger the data buffers, the greater the lag between computation, and visualization. Increasing the buffer sizes, however, reduces the system CPU load, and overall network traffic because larger more efficient packets are used at the transport level, and less I/O waiting occurs. In general, these questions must be answered by the application programmer. Given a value for an acceptable time delay, the application programmer must know the time required to compute each data point (highly application specific) in order to determine the number of data points that can be buffered without violating the acceptable time delay criteria. In more complex visualization models than were shown here, the time delay from network transport to physical rendering must also be added, and the buffer sizes reduced accordingly.

In the Lgraph example, network transport and rendering times can be neglected with respect to the computational time requirements if the transfer rate is less than approximately 1000 data points (one X and 2 Y values) per second.

In the Image example, the network transport times cannot be neglected due to the large transfers that must take place to render the associated images. This lag must be computed depending on the limits of the physical network connection. Since the data received is close to the form necessary for display, negligible time is required to render the image once the data is in the workstation. Since the memory requirements per individual frame are high for even modest frame sizes, it is doubtful, however, that multiple frames could be efficiently buffered.

SP was designed to perform near real-time scientific visualization. For this reason, buffer sizes are set by default to give performance as close to real-time as possible [26].

Resource Allocation

SP and X differ widely on system resource allocation. SP requires that all system resources be used once, and then disposed. The X Window System, however, allocates and holds system resources until the user has completed all manipulations. The difference is as follows, where X sends a series of drawing commands to render a display, SP sends all of the scientific data that makes up the display. In many cases, the scientific data is slightly larger than the associated drawing commands, but once sent, never needs to be resent. Similar conclusions can be drawn from memory and CPU resources for similar reasons.

X does not depend on the attached workstation for anything more than display rights. SP utilizes the workstation as a tool with a life longer than the computational problem. This point cannot be emphasized enough.

Performance Considerations

In order to achieve high overall performance of a distributed scientific visualization paradigm of any type, several important issues must be addressed.

On the workstation, the following problems must be dealt with: communication latency, and buffering. The latency in processing incoming data can have a large impact on the operation of the distributed system as a whole. The higher latency times of the Lgraph example drove the cost of network communication up to 5 microseconds of Cray CPU per byte of network data transfer, because of the overall reduction in channel capacity and resulting I/O wait scheduling problems on the supercomputer. Buffering requirements are also related to latency times. Larger receive buffers on the workstation can mitigate slower processing provided the data set is small enough. Large data sets can quickly destroy this proposition.

On the supercomputer, the following implementation problems must be dealt with: vectorization, and the maximum rated speed of the communications channel. In order for a distributed scientific visualization paradigm to be worthwhile, the supercomputer application must take advantage of the machine architecture and achieve the highest possible performance. This is the primary reason for a distributed visualization paradigm. Problems do occur, however, in cases where large amounts of data are generated faster than the entire visualization pipeline can handle it. Supercomputer system CPU usage increases dramatically due to I/O wait scheduling in configurations where the channel is extremely overloaded. In this case, additional buffering does not help. A knowledge of channel capacity can help the application programmer properly tailor the data requirements of the application to a level that can consistently reduce supercomputer system overhead.

Final Conclusions

In comparison to the closest available alternative, the X Window System, this distributed scientific visualization paradigm delivers what it promises: better supercomputer CPU and memory utilization, and decreased network loading. In some cases, CPU requirements can be reduced by an order of magnitude, while memory requirements are reduced to 1/3. Network utilization and interactive performance are highly dependent on the application, but can also improve significantly.

REFERENCES

1. Ackerman, Mark, Widget Writing, Xhibition '90, MIT X Consortium, San Jose, CA, May 1990.

2. Barton, Jim, Distributed Visual Processing, Silicon Graphics, Inc. Presentation, SIGGRAPH '90, Silicon Graphics, Inc., Dallas, TX, August 1990.

3. Brunhoff, Todd, VEX (Video Extension to X) Your Hardware, Xhibition '90 Conference Proceedings, Tektronix, Inc., San Jose, CA, May 1990, pp. 31-36.

4. C Programmer's Guide, Sun Microsystems, Inc., Mountain View, CA, 1989, pp. 78-82.

5. Chemtool, Cray Research, Inc., Minneapolis, MN, 1990.

6. Clare, Fred, NCAR Graphics User Guide Version 2.00, National Center for Atmospheric Research, Boulder, CO, August 1987.

7. Cray Y-MP Functional Description Manual, Cray Research, Inc., Minneapolis, MN, 1989, pp. 2.12-2.21.

8. DI-3000 User Guide, Precision Visuals, Boulder, CO, 1989.

9. Dyer, D. Scott, apE Providing Visualization Tools for a Statewide Supercomputing Network, Proceedings of the 24th Semi-Annual Cray Users Group Meeting, Columbus, OH, September 1989, pp. 237-241.

10. Foley, J.D., and Dam, A. Van, Fundamentals of Interactive Computer Graphics, Addison Wesley, New York, NY, 1982.

11. Fraser, M. D., Galiano, R. A., and Schaefer, M. E., Modeling the Cost of Resource Allocation in Distributed Control, Simulation Digest, Nashville, TN, Vol. 20, No. 4, April 1990, pp. 151-164.

12. Guitard, R. and Ware, C., A Color Sequence Editor, ACM Transactions on Graphics, New York, NY, Vol. 9, No. 3, July 1990, pp. 338-341.

13. Maillot, Patrick G., XGL Architecture, SunTech Journal, Peterborough, NH, Vol. 3, No. 2, May 1990, pp. 10-22.

14. Marshall, Robert, Visualization Methods and Simulation Steering for a 3D Turbulence Model of Lake Erie, Proceedings of the 1990 Symposium on Interactive 3D Graphics, Computer Graphics, Los Alamitos, CA, Vol. 24, No. 2, March 1990, pp. 89-97.

15. Morse, Alan, Visualizing 'Near Real-Time', SunTech Journal, Peterborough, NH, Vol. 3, No. 3, July 1990, pp. 52-56,60-62.

16. Network System Calls Reference Manual, Cray Research, Inc., Minneapolis, MN, 1990.

17. Ohio Supercomputer Center Graphics Project, apE 2.0 Technical Info, Ohio Supercomputer Center, Columbus, OH, March 1989.

18. O'Reilley, Tim, The Xlib Programming Manual, O'Reilley & Associates, Sebastopol, CA, 1990.

19. O'Reilley, Tim, The Xlib Reference Manual, O'Reilley & Associates, Sebastopol, CA, 1990.

20. O'Reilley, Tim, The Xt Intrinsics Programming Manual, O'Reilley & Associates, Sebastopol, CA, 1990.

21. O'Reilley, Tim, The Xt Intrinsics Reference Manual, O'Reilley & Associates, Sebastopol, CA, 1990.

22. Packard, Keith, Server Internals, Xhibition '90, MIT X Consortium, Cambridge, MA, May 1990.

23. Reed, M. W., Distributed Simulation Using Distributed Control Systems, Simulation Digest, Nashville, TN, Volume 20, No. 4, April 1990, pp. 143-150.

24. Rew, Russel, K., netCDF User's Guide, Unidata Program Center, Boulder, CO, January, 1990.

25. Scheifler, R.W. and Gettys, J., The X-Window System, ACM Transactions on Graphics, New York, NY, Vol. 5, No. 2, April 1986, pp. 79-109.

26. Shelef, D. Gary, Developing Graphics in Real-time, SunTech Journal, Peterborough, NH, Vol. 3, No. 3, July 1990, pp. 64-74.

27. SunOS 4.1 Network Programming Manual, Sun Microsystems, Mountain View, CA, 1990.

28. SunOS 4.1 Reference Manual, Sun Microsystems, Inc., Mountain View, CA, 1990.

29. Thompson, John M., Advanced Scientific Visualization Methods For Analysis of Technical Data, SunTech Journal, Peterborough, NH, Vol. 3, No. 3, July 1990, pp. 76-86.

30. UNICOS User's Guide, Cray Research, Inc., Minneapolis, MN, 1990.

31. UNICOS Performance Monitoring Utilities, Cray Research, Inc., Minneapolis, MN, 1990.

32. Voorhies, Douglas, David Kirk, and Olin Lathrop, Virtual Graphics, SIGGRAPH '88, Computer Graphics, Los Alamitos, CA, Vol.22, No. 4, August 1988, pp. 247-253.

33. VX/MVX Technical White Paper, Sun Microsystems, Inc., Mountain View, CA, 1990.

APPENDIX A
VISUALIZATION WIDGETS

Introduction

The visualization widgets provide the low level rendering functionality needed by SP. They are implemented as X11 R4 "widgets" to facilitate their reuse in other areas as well as to simplify the comparison of a direct X11 implementation vs SP. Note that these widgets were subclassed from Core. Therefore, few modifications are necessary to compile and run them under any Xt based widget scheme. Versions of these widgets have successfully run under the R4 Athena Widgets (Xaw), the R3 Athena Widgets, and the Open Look Intrinsic Toolkit (OLIT). There are two visualization widgets: the Lgraph widget, and the Image widget. The Lgraph widget turns X-Y scientific data into a line graph, and the Image widget turns raster data into a color image.

Lgraph Widget

A widget to display X-Y datasets.

Synopsis

```
#include <X11/StringDefs.h>
#include <X11/Intrinsic.h>
#include <Xvw/Lgraph.h>
widget = XtCreateWidget(widget, lgraphWidgetClass, . . .);
```

Class Hierarchy

Core->Lgraph

Description

The Lgraph widget is used to display X-Y data in a line graph form. It receives input from the internal function call LgraphAddData, and plots it. A single callback is supplied (specified by XtNcallback) to support additional functionality defined by the programmer. There are a number of display options including the number and size of tick marks, gridlines, and colors.

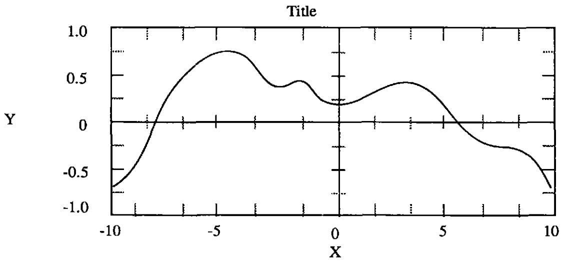

Fig. A.1. Line graph (Lgraph) widget

Resources

When creating an Lgraph widget, the following resources are retrieved from the argument list or from the resource database:

Resource	Type	Default	Description
XtNwidth	int	400	Total width of widget in pixels
XtNheight	int	400	Total height of widget in pixels
XtNx	int	None	X position of widget relative to parent
XtNy	int	None	Y position of widget relative to parent
XtNxdividers	Boolean	FALSE	Division lines along the X axis
XtNydividers	Boolean	FALSE	Division lines along the Y axis
XtNxdivisions	int	5	Number of tick marks/lines along X axis
XtNydivisions	int	4	Number of tick marks/lines along Y axis
XtNxminor	int	5	Number of minor tick marks between major tick marks on the X axis
XtNyminor	int	5	Number of minor tick marks between major tick marks on the Y axis
XtNticksize	int	10	Size in pixels of the major tick marks
XtNmticksize	int	6	Size in pixels of the minor tick marks
XtNxlogscale	Boolean	FALSE	Compress x to log scale
XtNylogscale	Boolean	FALSE	Compress y to log scale
XtNforeground	Pixel	Black	Foreground color

XtNbackground	Pixel	White	Background color
XtNcolor0	Pixel	Black	Color of trace 0
XtNcolor1	Pixel	Red	Color of trace 1
XtNcolor2	Pixel	Green	Color of trace 2
XtNcolor3	Pixel	Blue	Color of trace 3
XtNcolor4	Pixel	Yellow	Color of trace 4
XtNcolor5	Pixel	Magenta	Color of trace 5
XtNcolor6	Pixel	Orange	Color of trace 6
XtNcolor7	Pixel	Gray	Color of trace 7
XtNshiftMode	int	10	Percentage to jump on overshoot
XtNtitle	char *	NULL	Text printed on top of graph
XtNstring	char *	NULL	Internal holder for non-printing title
XtNxlabel	char *	"X"	X axis label
XtNylabel	char *	"Y"	Y axis label
XtNlabelFont	char *	vtsingle	Text font to use on labels
XtNfont	char *	vtsingle	Text font to use on numeric tick markers

The new resources (not inherited from superclasses) associated with the Lgraph widget are:

XtNxdividers	Specifies the existence of division lines in the vertical direction instead of tick marks on the X axis.
XtNydividers	Specifies the existence of division lines in the horizontal direction instead of tick marks on the Y axis.
XtNxdivisions	Represents the number of areas the X axis will be divided into.
XtNydivisions	Represents the number of areas the Y axis will be divided into.
XtNxminor	Represents the number of minor tick marks between each major division (see XtNxdivisions) on the X axis.
XtNyminor	Represents the number of minor tick marks between each major division (see XtNydivisions) on the Y axis.
XtNticksize	Represents the length in pixels of the major division tickmarks (see XtNxdivisions and XtNydivisions.)
XtNmticksize	Represents the length in pixels of the minor division tickmarks located between the major division tickmarks (see XtNxminor

	and XtNyminor.)
XtNxlogscale	Specifies that the X axis should be log scale (currently not implemented.)
XtNylogscale	Specifies that the Y axis should be log scale (currently not implemented.)
XtNforeground	Specifies the default foreground color for axis and lables.
XtNbackground	Specifies the default background color used for everything displayed in an Lgraph widget.
XtNcolor[0-7]	Specifies the color for each data trace on the graph. If there are more than 8 traces, colors will be recycled (mod 8.)
XtNshiftMode	Represents the percentage that the graph should extend the minimum or maximum limits by if real-time data exceeds the current minimum or maximum values.
XtNtitle	Specifies the text that will be placed in the center top portion of the graph to label it. This text will be displayed using XtNlabelFont.
XtNstring	Specifies text that will not be displayed anywhere, but is useful to keep track of internal parameters.
XtNxlabel	Specifies text that will be displayed under the X axis using the XtNlabelFont.
XtNylabel	Specifies text that will be displayed to the left of the Y axis using the XtNlabelFont.
XtNlabelFont	Specifies the font that will be used in displaying labels.
XtNfont	Specifies the font that will be used in displaying numeric data.

Translations and Actions

The following are the default translation bindings that are used by the Lgraph widget:

```
<Btn1Down>:          notify()
```

The Lgraph widget supports the following actions:

Processing application callbacks (XtNcallback) with notify().

Programmatic Interface

To create an Lgraph widget instance, use XtCreateWidget and specify the class variable lgraphWidgetClass.

To destroy an Lgraph widget instance, use XtDestroyWidget and specify the widget ID of the Lgraph widget.

To add a data point to the Lgraph widget's database, use lgraphAddData, and specify the widget ID of the Lgraph widget, and the data that should be added and displayed.

```
lgraphAddData(w, x, yv, yvc)
LgraphWidget w;
double x;
double yv[];
int yvc;
```

The parameter w is an Lgraph widget created with XtCreateWidget. The x parameter is the x value of the X-Y data. The yv[] (y vector) is an array of y values that will be associated with the x value. The yvc (y vector count) is the length of the yv[].

To modify the Lgraph widget's graphing limits, use lgraphSetLimits, and specify the widget ID of the Lgraph widget, and the new X, and Y limits. Note that the limits are not specified as resources! They must be set using lgraphSetLimits, otherwise they default.

```
lgraphSetLimits(w, minx, miny, maxx, maxy, mode)
LgraphWidget w;
double minx, miny, maxx, maxy;
int mode;
```

The w parameter is an Lgraph widget created with XtCreateWidget. The min and max parameters are the display limits for the x and y axis. These are not hard limits. If a limit is exceeded, the Lgraph widget will expand the axis depending on the mode. The mode is the percentage that the limits will expand by if they are exceeded. It is an integer values, so for example, a mode of 10 means that the display will be increased by 10% if the limits are exceeded.

Image Widget

Convert field data to a color image for display.

Synopsis

```
#include <X11/Intrinsic.h>
#include <Xvw/Image.h>
widget = XtCreateWidget(widget, imageWidgetClass, . . . );
```

Class Hierarchy

Core->Image

Description

The Image widget converts a field of scalar data into a color raster display. One hundred and twenty eight colors are supported (7 bits). Input data is provided either in the form of a full field linear array of 8-bit values (the topmost bit is truncated - reserved for future use), or a delta form.

The image data of an ImageWidget is stored in one of two forms, either a pixmap, or an XImage. The pixmap is resident on the display server, is fast to display, but slow to change. The XImage is resident in the client, is fast to change, but slow to display. All of the scientific data formats use XImages, but the pixmap interface is there for convenience. There is no resource value for an initial XImage, it's size must be specified as XtNwidth, and XtNheight, and its value with an ImageSet command once the widget has been created. One additional problem occurs with pixmaps: they are incompatible between different depths. In other words, don't try to display a bitmap (depth = 1) on a color display (depth > 1), because it won't convert properly, and a Bad Match error will result. This problem does not occur with XImages.

Resources

When creating an Image widget, the following resources are retrieved from the argument list or from the resource database:

Resource	Type	Default	Description
XtNwidth	int	400	Total width of widget in pixels
XtNheight	int	400	Total height of widget in pixels
XtNx	int	None	X position of widget relative to parent
XtNy	int	None	Y position of widget relative to parent
XtNbitmap	Pixmap	None	Image to display
XtNcallback	Callback	None	Action translation
XtNforeground	Pixel	Black	Foreground color
XtNbackground	Pixel	White	Background color

The new resources (not inherited from superclasses) associated with the Image widget are:

XtNbitmap	Specifies an image to be displayed in the window. If XtNwidth and XtNheight are not specified, the XtNbitmap is queried for width and height, and XtNwidth and XtNheight are set accordingly.
XtNcallback	Specifies the callback interface to the notify() action.

XtNforeground	Specifies the default foreground color (an Image widget may be blank.)
XtNbackground	Specifies the default background color used for everything displayed in an Image widget.

Translations and Actions

The following are the default translation bindings that are used by the Lgraph widget:

```
<Btn1Down>:          notify()
```

The Image widget supports the following actions:

Processing application callbacks (XtNcallback) with notify().

Programmatic Interface

To create an Image widget instance, use XtCreateWidget and specify the class variable imageWidgetClass.

To destroy an Image widget instance, use XtDestroyWidget and specify the widget ID of the Image widget.

To rapidly change the current pixmap that is displayed by the widget, use the ImagePixmapSet command.

```
ImagePixmapSet(w, newmap)
ImageWidget w;
Pixmap newmap;
```

Ordinarily, the current pixmap is changed by setting XtNbitmap with the XtSetValues command. This implies, however, an XClearWindow call to first clear the window before the new XtNbitmap is displayed. Using ImageSet gets around this problem, and sets the resource value directly, handling all the necessary screen I/O without going through the normal Xt process. If the new pixmap is a different size than the old one, ImagePixmapSet will make a geometry request and alter its size to fit the new XtNbitmap.

To rapidly change the current image that is displayed by the widget, use the ImageSet command.

```
ImageSet(w, image)
ImageWidget w;
XImage *image;
```

Images are the display independent, client resident version of a pixmap. They are faster to manipulate locally, but require some overhead to transfer in their entirety. Using the ImageSet command will rapidly change the currently displayed image to a new value.

The Image widget also supports a 7-bit scientific data format. One hundred twenty eight colors are allocated when the widget is created. Most color screens are 8-bit, allowing 256 colors, however, allocating the full color set would preclude other applications, and mess up currently running applications. For this reason, a 7-bit implementation was chosen. The data format for the scientific data format is simply an array of 8-bit unsigned integers (unsigned char). The width and height of the picture is assumed to be the same as the current width and height of the widget. Colors are allocated with the least dense color as zero, and the most dense color as 127. Therefore, to represent a scalar field, the color of a point is simply:

$$Color = \frac{127X}{X_{Max} - X_{Min}}$$

The ImageScientificSet command is used to convert the 7-bit array of data to a pixmap that can be displayed. The format of the ImageScientificSet command is as follows:

```
ImageScientificSet(w, data)
ImageWidget w;
unsigned char data[];
```

Where the size of the array data[] is determined by XtNwidth * XtNheight. Once it receives data, and converts it to Pixmap form, ImageScientificSet calls ImageSet to display the data, and properly set the current Pixmap (XtNbitmap).

The Image widget can also accept data in the scientific delta format. This format relies upon the existence of a correct current Image, and relays only the changes that need to be made to the current Image to correct it. Data is sent in a buffer, similarly to the 7-bit scientific data format, with the exception that it is segmented. The format consists of a series of 8-bit values:

Command, Length, Data, Data, . . . , Data, Command, Length, Data, ...

Command is a single 8-bit field that is either 'S' for skip, 'U' for use, or 'D' for done. The Length is an 8-bit (0-255) unsigned integer that tells how many data bytes follow for a use command, or how many bytes to skip for the skip command. The Data area is only present for the 'U' command, and represents data that should be copied into the new pixmap from the old. The 'D' command is not followed by a length, and signifies the end of the buffer. For more information about delta format, see Appendix D.

The scientific delta format permits rapid rendering over a potentially slow network connection. The command for processing scientific delta format buffers is ImageDeltaSet.

```
ImageDeltaSet(w, delta)
ImageWidget w;
unsigned char delta[];
```

The end of the buffer is designated by the 'D' command, so no length parameter is required. The ImageDeltaSet command calls ImageSet once it has computed the new image.

APPENDIX B
SP COMMUNICATIONS LIBRARY

Introduction

Libsp is the programming interface to the distributed graphics rendering package SP. Using SP to render graphics is much simpler than using full blown graphics packages like DISSPLA or DI-3000, as it has fewer display features, but it has the benefit of offloading graphics processing to the graphics display server. This feature allows highly vectorized supercomputer programs to run more efficiently than they could otherwise. Furthermore, the libsp library is much smaller than full-blown graphics packages, allowing the application more memory room than is otherwise available.

Programs that use libsp to render graphs are structured into five basic steps: initialize the communication channel, initialize an object instance, write data to the object, close the object, and shutdown the communication channel.

To write a program that uses libsp, include the header file <sp.h>, and link with the libsp library as in the following example:

```
cc -o demo demo.c -lsp -lm
```

To run a program that uses libsp, first start the sp daemon on the machine that will be doing the display work. Next, tell the computational program which sp daemon to connect to by either setting the environment variable DISPLAY (with the traditional X Window System notation), or with the -display option on the command line. Any of the following examples will work correctly:

Example 1 (command line server resolution):

```
demo -display adrastea.tamu.edu
demo -display adrastea.tamu.edu:0
demo -display 192.58.110.60
demo -display 192.58.110.60:0
```

Example 2 (environment variable server resolution):

```
setenv DISPLAY adrastea.tamu.edu
demo
```

General Purpose

The general purpose routines are common to all of the SP display models. Their primary concern is to initialize the communications channel and shut it down in an orderly fashion when all communications are done.

SpInitialize

Open a communication channel to an SPD on another computer.

```
#include <sp.h>

Display SpInitialize(argc,argv)
(int *)argc;
(char *)argv[];
```

SpInitialize() receives as input the command line parameters. It parses the command line looking for display options. If any options are found that it understands, they are removed from the argc, argv list. Any options SpInitialize finds that it does not understand are left intact. Allowable display options are:

-display *address*

Where *address* is either the name or TCP/IP address in dot notation of a workstation running the sp display daemon. libsp also looks in the environment variable DISPLAY for a default address to connect to. If the -display option is used, that address will be used over the environment variable DISPLAY. Note that SP will take either the standard X notation *address:screen* or just *address*.

SpShutdown:

Stop the SP communications channel in an orderly fashion.

```
#include <sp.h>

void SpShutdown(display)
(Display)display;
```

SpShutdown() is called when all graphics data has been sent, and the program is ready to quit. It is not absolutely necessary for SpShutdown() to be called when a program exits, as any currently open network sockets will be closed by UNIX. Calling SpShutdown() before exit(), however will reduce the number of errors on the server side, and is considered to be the definitive way to stop an SP graphics driven program.

Line Graphs

Line graphs display data as their name implies. Data sets are made up of groups of X-Y data. SP assumes that data is given to it in the order that the lines should be connected. No curve fitting is done, the graphs are simple connections of data points. An example of a program that uses the line graph display model is shown below.

```
/*
  DEMO.C

  Change History:
      6-Jul-90/mwl - Original Issue
*/
#include <math.h>
#include "sp.h"

main(argc,argv)
int argc;
char *argv[];
{
  Display display;
  Graph graph;
  float x;
  float max;
  float yv[2];
  int i;

  if ((display = SpInitialize(&argc,argv)) < 0)
  {
    printf("Error: Can't open graph\n");
    exit(0);
  }

  if (argc > 1)
    max = atof(argv[1]);
  else
    max = 5.0;

  if ((graph = SpLgraphOpen(display,0,-1.0,max,1.0,"Angle","Value","Sin X")) <0)
  {
    printf("Error: Can't open graph\n");
    exit(0);
  }

  SpLgraphGridlines(display,graph,TRUE,TRUE);
  SpLgraphDivisions(display,graph,IGNORE,10);

  for (x = 0.0; x < max; x += 0.01)
  {
    for (i = 0; i < 20000; i++)
    {
      yv[0] = sin(x);
      yv[1] = cos(x);
    }
    SpLgraphAddBinaryData(display,graph,x,yv,2);
  }

  SpLgraphClose(display,graph);

  SpShutdown(display);
}
```

SpLgraphOpen

Create an instance of a line graph (Lgraph).

```
#include <sp.h>

Graph SpLgraphOpen(display,minX,minY,maxX,maxY,xlabel,ylabel,title);
(Display)display;
(double)minX;
(double)minY;
(double)maxX;
(double)maxY;
(char *)xlabel;
(char *)ylabel;
(char *)title;
```

The display parameter is a value returned from SpInitialize(). The xlabel, ylabel, and title can be assigned NULL to let their values default.

SpLgraphAddData

Add a data point to an already instantiated Lgraph.

```
#include <sp.h>

void SpLgraphAddData(display,graph, x, yv, yvc)
void SpLgraphAddBinaryData(display,graph, x, yv, yvc)
(Display)display;
(Graph)graph;
(float)x;
(float *)yv;
(int)yvc;

void SpLgraphAddDataD(display,graph, x, yv, yvc)
(Display)display;
(Graph)graph;
(double)x;
(double *)yv;
(int)yvc;
```

SpLgraphAddData is the most generic way to display data generated on any architecture. It uses the ASCII protocols built into SPD to communicate. SpLgraphAddBinary-Data converts all data to IEEE floating point formats before sending it to SPD using the binary communication protocols. SpLgraphAddBinaryData is currently only supported on

Cray Y-MPs, but when used on that platform, it is about five times faster than SpLgraphAd-dData.

SpLgraphAddDataD is the double version of SpLgraphAddData. It takes as parameters doubles instead of floats. Its use is not recommended, because it also uses the ASCII protocols, and is very slow.

The graph parameter is a value returned from SpLgraphOpen(). Each point on a graph is a combination of an X value, and one or more Y values. The X value is stated in the parameter x, and the multiple Y values are stated in the parameter yv (Y-Vector). The Y values are in vector form (an array of doubles), whose length is determined by the parameter yvc.

SpLgraphGridlines

Turn gridlines on (instead of tickmarks) for an already instantiated Lgraph.

```
#include <sp.h>

void SpLgraphGridlines(display,graph,xgridlines,ygridlines)
(Display)display;
(Graph)graph;
(Boolean)xgridlines;
(Boolean)ygridlines;
```

The xgridlines and ygridlines parameters specify whether lines or just tick-marks will be drawn on the graph at the x and y axis label values. A value of TRUE, (non-zero) specifies that lines will be drawn. A value of FALSE (zero) specifies that only tick-marks will be drawn.

SpLgraphDivisions

Set the number of divisions on the X and Y axis for an already instantiated Lgraph.

```
#include <sp.h>

void SpLgraphDivisons(display,graph,xdivisions,ydivisions)
(Display)display;
(Graph)graph;
(int)xdivisions;
(int)ydivisions;
```

The xdivisions, and ydivisions parameters specify the number of tick marks on the x and y axis that will be displayed. If either value is set to IGNORE, then it will default.

SpLgraphClose

Close the connection to an Lgraph. After the connection is closed, no further data can be added to the Lgraph.

```
#include <sp.h>

void SpLgraphClose(display,graph)
(Display)display;
(Graph)graph;
```

SpLgraphClose() is called when all data to a particular graph has been sent. The graph parameter is a value returned from SpLgraphOpen().

Images

Images are used to represent two dimensional matrices. Once the width and height are set, SP assumes that it will stay constant for the duration of the connection. For an image of width WIDTH, and height HEIGHT, access should be done in the following manner. To set a value at point (x,y), use:

```
image[x + y * WIDTH] = value;
```

Where value is an unsigned char (0-255). Due to color palatte and processing constraints, only one image may be open at a time. An example that uses that image display model is shown below.

```
#include "sp.h"

#define WIDTH 400
#define HEIGHT 400
#define COLORS 40

unsigned char data[500 * 500];

main(argc,argv)
int argc;
char *argv[];
{
  Display display;
  int width,height;
  Image image;
  int i,x,y;
  register int j;

  display = SpInitialize(&argc,argv);

  if (argc > 1)
  {
    width = atoi(argv[1]);
    height = width;
  }
  else
  {
    width = WIDTH;
    height = HEIGHT;
  }

  image = SpImageOpen(display,width,height,"blue", "red", COLORS);

  printf("Animating....\n");

  /* provide some animation */
  for (i = 0; i < height; i += 4)
  {
    for (j = 0; j < width * i; j++)
      data[j] = COLORS - 1;
      for (; j < width * height; j++)
        data[j] = 0;

      for (y = i; y < height && (y  < (i + COLORS * 3)); y += 3)
      {
        for (x = 0; x < width; x++)
        {
          data[x + y * width] = COLORS - (y - i) / 3 - 1;
          data[x + (y + 1) * width] = COLORS - (y - i) / 3 - 1;
          data[x + (y + 2) * width] = COLORS - (y - i) / 3 - 1;
        }
      }

      SpImageAddData(display,image,data);
    }

    SpImageClose(display,image);

    SpShutdown(display);
  }
```

SpImageOpen

Create an instance of an Image.

```
#include <sp.h>

SpImageOpen(display,width,height,lowcolor,highcolor,numcolors)
Display display;
int width;
int height
char *lowcolor;
char *highcolor;
int numcolors;
```

An image is specified as a matrix with a particular color mapping. Colors are specified as a highcolor, a lowcolor, and an integer number of color incrementes (numcolors) used to fade between the high and low. When in operation, the values 0 to numcolors - 1 will map to the lowcolor faded to the highcolor respectively. Values over numcolors - 1 will map to the highcolor. The lowcolor and highcolor are represented as ASCII strings like "red" or "blue". The only consideration required when specifying colors is that they must be in the color database of the X server providing display service to the SPD. This information can be found by looking in the /usr/lib/X11/rgb.txt file on the computer that is running SPD. If a color mapping failes, it will default to the default foreground color of the SPD. This scheme was chosen to give the application programmer as much control as possible over the color palatte to be represented.

SpImageAddData

Send a frame of image data to be displayed by the SPD.

```
#include <sp.h>

SpImageAddData(display,image,data)
Display display;
Image image;
char *data;
```

The buffer (char *)data is the matrix of values mapped to the appropriate colors as specified in SpImageOpen. It is assumed to be of width and height specified in the SpImageOpen. If for some reason the width and height change, erroneous results will occur.

SpImageClose

Closes an image connection.

```
#include <sp.h>

SpImageClose(display,image)
Display display;
Image image;
```

SpImageClose returns nothing, but must be executed before another image can be opened.

APPENDIX C
CRAY TO IEEE DATA CONVERSION

Introduction

Cray Research, Inc. uses nonstandard formats for all floating point data. In order to write binary data files on a Cray that can be later read on another machine, conversion routines must be used. UNICOS does not currently supply subroutines that can convert Cray floating point formats to the IEEE standard formats that can be read by most computers, so the programmer is left to do it alone. These conversion routines are expected in the next release of UNICOS.

IEEE Format

Most computers use ANSI IEEE 754-1985 standard floating point formats to store numeric data[1]. Under this standard, single precision numbers are 32 bits in length, and double precision numbers are 64 bits in length. Each number is made up of three parts: a sign bit, mantissa, and exponent, in the following format:

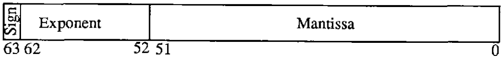

Fig. C.1. IEEE 64 bit floating point format

In the double precision form, the exponent is an eleven-bit value, biased by 1023, the Mantissa is a normalized fraction, and the sign bit is set when the value is negative. The single precision form is similar, except for the smaller bit fields. In general, floating point values in IEEE format can be written in the form:

$$value = (-1)^{Sign} 2^{(exponent - bias)} 1.f$$

Cray Research Format

Cray Research has a different definition for the storing floating point numeric data[2]. Since Crays are 64-bit per word machines, single precision is considered to be 64 bits in length, and double precision is 128 bits. Single precision on a Cray is considered double precision on IEEE format machines. Like the IEEE format, the Cray format consists of a sign bit, mantissa, and exponent in the following format:

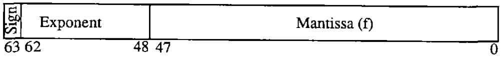

Fig. C.2. Cray 64 bit floating point format

Where the exponent is a 15 bit biased integer with a bias value of 40000_8, and the mantissa is a 48 bit signed fraction. Since the sign of the entire number is located in bit 63, the mantissa is an unsigned, normalized value. The actual value of a number stored in this format is represented as follows:

$$value = (-1)^{sign} 2^{(exponent - 40000_8)} 0.f$$

Cray to IEEE Conversion

Generating binary data files on a Cray for later processing on another machine is not possible unless the data is written in a form that can be read by the other machine. Since the 64 bit Cray floating point format is the most often used, and 64 bit data formats are available on a large number of smaller computers, an example showing the conversion of Cray 64 bit to IEEE 64 bit is given here.

To convert Cray 64 bit floating point numbers to IEEE 64 bit floating point, the conversion process proceeds as follows: First, the sign bit, mantissa, and exponent are removed from the original number, and appropriately denormalized. Next, the mantissa and exponent are renormalized into IEEE form, and all three parts are recombined. A small section of C code illustrates the process:

```
unsigned int crayFloatToIEEE(f)
float f;
{
    unsigned int i, *iptr,sign,exponent,coeff,new;

    /* special case at zero */
    if (f == 0.0)
        return (0x0000000000000000);

    /* break the old value down into it's components */
    iptr = (unsigned int *)&f;
    i = *iptr;
    sign = i & 0x8000000000000000;
    frexp(f,&exponent);

    /* modify the coefficient so it fits the 1.f format */
    coeff = ((i & 0x0000ffffffffffff) << 1) & 0x0000ffffffffffff;
    exponent--;

    /* rebias and check for exponent overflow */
    exponent += 1023;

    if (exponent > 0x7fe || exponent < 1)
        return (0x7ff0000000000001); /* not a number */

    /* put the new value together from scratch */
    new = sign | (coeff << 4);
    new |= (exponent) << 52;

    return (new);
}
```

Special care must be taken to handle the three overflow possibilities. Zero is a special case and must be detected and handled by itself. Since the Cray format has a larger exponent than IEEE, an overflow can occur in either the positive or negative direction. In this case, there is a bit shifted out of the 10 bit wide IEEE format exponent when an overflow occurs. The response to an overflow condition can be varied, but ususally implies either storing the value 0x7FF0000000000000 (+ Infinity) for positive overflows, or 0xFFF0000000000000 (- Infinity) for negative overflows as given in the IEEE standard, or returning 0x7FF0000000000001 (not a number).

The use of bitwise operators is fast and efficient. Note that the data is stored in an unsigned int value (64 bit). The unsigned int is used in this case as an 8 byte data buffer. A write operation to a file with the address of the unsigned int as an argument will properly store the value on disk:

```
write(handle,(char *)&value,8);
```

Although not explicitly necessary, the char * typecast relieves warning messages in most compilers.

Conclusion

Converting IEEE to Cray format is simply the reversal of the Cray to IEEE algorithm. The same type of process applies: first extract the sign bit, exponent, and mantissa, next manipulate the mantissa and exponent into their new forms, and finally put them back together into the proper bit fields.

Using binary floating point format values has several benefits. First, storage requirements are dramatically reduced. The Cray 64 bit format gives approximately 15 decimal digits of precision over a decimal range of 10^{-2466} to 10^{+2466}. To store numbers of this magnitude in ASCII form would require about 20 bytes each instead of the binary 8. Similar space requirements are seen in IEEE versus ASCII formats. The next most important benefit is speed. It is much faster to write eight bytes (64 bits) of binary data than it is to convert that same eight bytes to ASCII, then output 20 bytes. Any postprocessing done on the data can also be dramatically improved. The slower I/O channels of smaller computers become less of a problem, and the CPU has fewer tasks to perform when files are read in binary form rather than ASCII.

APPENDIX D
DELTA COMPRESSION

Few computer/network systems have the necessary bandwidth to quickly render sequences of digitized images. In order to get a reasonable frame rate (>1/5 Hz) with current technology, the image information must either be compressed, or special dedicated hardware must be present. The SP image example uses a simple form of image sequence compression called delta compression. With delta compression, an image is assumed to be part of a sequence of regularly ordered images. Once the first image is transferred, only the changes necessary to convert the old image to the new image actually need to be transmitted.

Theoretical Performance

Providing a reasonable number of pixels change per frame, large savings can occur in terms of network loading. Obviously, the performance of such a scheme is extremely dependent on the application. The SP image application was designed to show scalar fields moving with time. From this context, it is reasonable to assume that as the boundaries of the iso-lines move, pixels will most often change in groups of less than 256, but greater than 1 (these are not hard values, just guesses). Given an overhead of 2 bytes per packet, plus data, if 25% of a 400 by 400 pixel image changes in each frame, and all the changes are single byte independent changes, the size of the delta compressed image would be .25*400*400*3+.75*400*400/256*4 or approximately 122 Kbytes. Not much savings over the 160 Kbytes just to transmit the whole frame. If the data were more reasonable, however, and on the average had just 32 bytes per frame, the delta buffer would become .25*400*400*34/32+.75*400*400/128*2, or approximately 44 Kbytes. A much more reasonable number, one fourth of the original size. In this case, with the network transmission time considered the limiting factor of the process, almost four frames could be transmitted and rendered in the time required to move one in the whole case. For cases where even larger average change packet lengths occur, even more compression would result.

SP Delta Format

The delta format used by SP consists of a sequence of commands to either use or discard data, stored in a linear buffer. All commands are relative to the current position in the image buffer that is updated after the execution of each command. There are three commands: use, skip, and done. The use command specifies a sequence of data bytes that will replace values in the current image, requires a variable number of bytes (unsigned char), and takes the following form,

U,length,data,data,data,data,data,data

The character U indicates the command use, the length value is an 8-bit unsigned number that indicates the number of data bytes that follow. The skip command is a two byte sequence that specifies the number of bytes to skip without changing in the original image. It has the following format,

S,length

The character S indicates a skip command, and the length is an 8-bit value indicating the number of bytes to skip over in the current image. The done command is a single byte command that indicates all processing is done for this frame. It has the following format,

D

The single D character indicates a done command.

General Comments

The SP delta format is not a generic compression algorithm. It makes several assumptions about well behaved data to get good performance characteristics. If SP were to be used for a specific application, it would be advisable to investigate the nature of the data to be moved, and optimize the compression algorithm appropriately.

One of the good points about the SP delta compression algorithm is that is requires relatively little CPU to get good results. The compression/decompression sequence is simple, and does not limit the overall processing throughput.

APPENDIX E
COMMUNICATION PROTOCOLS

The SP network communications protocols are of key importance to the operation of SP as a whole. All data generated on the supercomputer is moved to the workstation via the SP communications protocols. There are two modes of operation within the protocol set: ASCII command mode, and binary mode. In both modes of operation, data packets are sent in newline terminated blocks. In ASCII command mode, the blocks are variable length, and in binary mode, the blocks are of fixed size.

Command Protocols

The command protocols are completed entirely in ASCII for ease in readibility as well as portability. All commands start with a # character, and are immediately followed by textual data, terminated by a newline character. All fields within an ASCII command can be separated with either spaces, commas, or tabs. If a single field contains multiple words separated with spaces, placing the entire field in double quotes will cause SPD to interpret the field as a whole rather than as several fields.

#shutdown

```
#shutdown\n
```

When received by SPD, this command turns off the communication channel. Although the communication channel can be closed without warning, the issuance of a #shutdown command eases any errors by providing an orderly shutdown.

Image Commands

There are two commands associated with image connections: #video and #vquit. The #video command opens an image sequence connection, and the #vquit command closes that connection.

#video

#video width height lowcolor highcolor numcolors\n

Where width and height relate to the size of the image associated with the data stream. Lowcolor is the color associated with the least intensity, highcolor is associated with the highest intensity, and numcolors is an integer value describing how finely to fade between the two. For more information on values to use, see the section on Libsp.

#vquit

#vquit\n

The #vquit command has no arguments, and stops the current image sequence connection.

Lgraph Commands

Lgraphs have more options than Images, and therefore support more commands. There are a total of nine Lgraph related commands: #lgraphm, #lgraph, #xdivisions, #ydivisions, #xgridlines, #ygridlines, #quit, #data, and default data.

#lgraphm

#lgraphm minx miny maxx maxy xlabel ylabel title\n

The #lgraphm command initializes an Lgraph instance in multiple connection mode. The min and max values are ASCII floating point numbers that are best guesses for the final values of the graph. It is not strictly important that they be correct, but the Lgraph will have an easier time showing the time progression of the data set if it doesn't have to constantly rescale to show the values. The labels and title are ASCII text strings that will be placed in the graph. If the labels and title are multi-word sequences separated with spaces, placing the values in quotation marks will allow SPD to properly group them. Once this command is sent, SPD will initialize and send back an lgraph connection id. The connection id will need to be used on all future Lgraph commands to associate data with a particular instance of an Lgraph.

#lgraph

#lgraph minx miny maxx maxy xlabel ylabel title\n

The #lgraph command has similar syntax to the #lgraphm command, however, Lgraphs opened in this fashion do not return a connection id. This is useful in single context situations where there can be no bi-directional communication. An example of this is file storage.

#xdivisions

#xdivisions value [graphno]\n

The #xdivisions command specifies the number of divisions along the X axis. The optional graphno parameter is the connection id returned by a call to #lgraphm. If no graphno parameter is specified, the #xdivisions command operates on the first graph on the connection id list.

#ydivisions

#ydivisions value [graphno]

The #ydivisions command operates similarly to the #xdivisions command on the Y axis.

#xgridlines

#xgridlines boolean [graphno]

The #xgridlines command changes the graph divider style between tickmarks and gridlines on the X axis (in the Y direction). The optional graphno parameter is the connection id returned from a call to #lgraphm. The absense of the graphno parameter indicates that the operation will be performed on the first graph in teh connection id list.

#ygridlines

#ygridlines boolean [graphno]

The #ygridlines command changes the graph divider style between tickmarks and gridlines on the Y axis (in the X direction). The optional graphno parameter is the connection id returned from a call to #lgraphm. The absense of the graphno parameter indicates that the operation will be performed on the first graph in the connection id list.

#quit

#quit [graphno]\n

The #quit command closes the connection id for the graph instance given by graphno. If graphno is not specified, the #quit command closes the first graph on the connection id list.

#data

#id x y1 y2 y3 ..... yn\n

The #data command adds a data point to the data set associated with a particular Lgraph instance. It has a slightly different format than the other # commands. The id field directly after the # character contains the connection id of the graph as returned by #lgraphm. The x and y values are formatted ASCII floating point numbers.

default data

If no # sign is present, the data is assumed to be in numeric (x y1 y2 .... yn) format, and will be associated with the first graph on the graph connection id list.

Binary Protocols

The binary protocols are used to transmit large blocks of data more efficiently than could be done through ASCII. Although binary blocks are newline terminated similarly to the ASCII command blocks, the newline character is only used as a frame check. If the newline character isn't where it should be, a framing error occured, and SP will report it as an unbound binary block.

Binary blocks are differentiated from command blocks with a different starting character. The starting characters for the binary blocks are '!', '@', and '$', while the starting character for the command blocks is '#'.

Image Format

There are two kinds of image blocks. Whole image blocks, and delta image blocks. The image protocol operates by first sending an entire image frame to establish a baseline, then sending a individual delta buffers to alter the previous image into the new image.

Image blocks are made up of four parts, a starting character, followed by a 64-bit integer length, the specified data, and a newline character. The length field specifies the number of bytes in the data block that follows. This value can be compared against known size of the image once the data is received by SPD, to determine the validity of the connection. The starting character for the image block is a '@' character.

Image delta blocks are made up of four parts also: a starting character, followed by a 64-bit length, the delta data, and a newline character. The length field specifies the number of bytes in the delta command buffer. For more information on the internal format of the delta buffers, see the section on delta compression. The starting character for an image delta block is a '$'.

Lgraph Format

The Lgraph binary buffer format is slightly more complex than the image binary block formats because multiple simultaneous channels across a single connection are supported. The Lgraph binary format is made up of the following sequence: a starting character '!', a channel id (8-bits), a Y vector length (yvc)(8-bits), an X value (64-bit IEEE floating point format), a sequence of yvc Y values (64-bit IEEE floating point format), and a newline character. In C structure form, the format is described as follows,

```
struct lgraphFormat
{
    char startingchar;
    unsigned char id;
    unsigned char yvc;
    unsigned char padding[6];
    double x;
    double y[yvc];
    char terminator;
}
```

All floating point numbers are in IEEE 64-bit floating point format. It is the responsibility of the machine doing the sending to ensure that all numbers are in a proper format. See the section on Cray to IEEE floating point format conversion for more details.

APPENDIX F

USING SP

The heart of the SP system is SPD. This is the software that the user interacts with most directly. The demonstration version has three parts: the SPD application window, the Lgraph window, and the Image window. All display models within SP have a particular type of window associated with them. As more display models are added, more window types with features specific to the display model will also be added.

Startup

Since SPD is a graphical application based on the X Window System, it must be connected to an X server. This can be done either remotely, or locally to the machine, but some models (the image model specifically) require high bandwidth between the application and the video display, so running them remotely will have a bad influence on their performance. Some models, like the Lgraph, require relatively small amounts of display bandwidth, and work fine run remotely over the network using the X protocols.

Usually, SPD is started with the command: sp -iconic &. Resulting in the following icon, stating that SP is ready to move data.

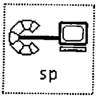

Fig. F.1. SP icon

Note that only one SPD can be run at a time. SP uses TCP port 10000 to communicate, and if it can't get that port in particular, will print an error message, and stop.

SPD Application Window

Expansion of the SP icon results in the display of the SP application window. For display models with a save option (like Lgraphs), old data sets can be reloaded from this window.

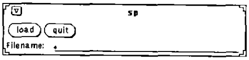

Fig. F.2. Application window

A file is loaded and displayed by typing in the filename next to the filename prompt, and clicking the load button. The quit button terminates all communication connections, and shuts SPD down.

Lgraph Window

An Lgraph window is created for every instance of an Lgraph display model connection. It has an area to render the associated X-Y line graph, and some buttons to deal with the data once it has been transferred to the Sun in its entirety.

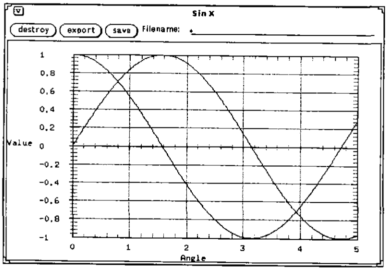

Fig. F.3. Lgraph window

The destroy button completely removes an Lgraph window from existence. All memory associated with the dataset is deallocated, and the window is closed. Any data associated with the Lgraph that was not saved before the destroy was instituted will be lost.

The export facility writes data in Xgraph form. To export data from an Lgraph into Xgraph form for further manipulation and printing, specify the filename to save the data to (usually a name .xg) after the filename prompt on the display and press the export button. All of the data associated with the Lgraph will be written to the specified file.

The save facility writes data in SP form so it can be retrieved later with the SP application window load feature. To save a file in SP form, specify its filename after the filename prompt in the Lgraph window, and press the save button. All of the data associated with the Lgraph will be written to the specified file.

Image Window

An image window is created for every instance of an Image display model. It contains an area to render color matrix information, and a single button to destroy the instance.

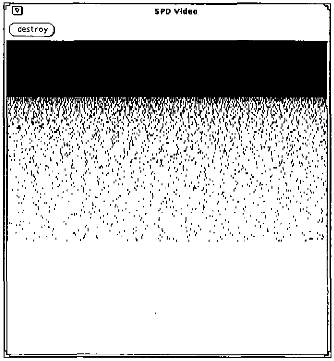

Fig. F.4. Image window

Due to processing and color palatte limitations, only one Image is allowed to be open and active at a time. Several Images can be open, but only one can actively receive data from the network.

APPENDIX G

SPD SOURCES

Source Code listings for SPD

```
/*
  SPD.H

  Change History:
      06-Jul-90/mwl - Original Issue
      03-Nov-90/mwl - Original Issue
*/

typedef struct graphappTag
{
  Widget shell;
  Widget graph;
  Widget text;
  int connected;
} GRAPHAPP;

typedef struct graphcontextTag
{
  struct graphcontextTag *nextgraph;
  Widget graph;
  int isrealized;
  int graphno;
  int connected;
} GRAPHCONTEXT;

typedef struct imagecontextTag
{
  Widget video;
  Widget videoShell;
  char *buffer;
  unsigned int buflength;
  int connected;
} VIDEOCONTEXT;

typedef struct commcontextTag
{
  GRAPHCONTEXT *lastgraph;
  GRAPHCONTEXT *firstgraph;
  VIDEOCONTEXT *video;
  int numgraphs;
} COMMCONTEXT;


#define SPD_TCP_PORT 10000
```

```
/*
  TOKENS.H

  Change History:
      06-Jul-90/mwl - Original Issue
      03-Nov-90/mwl - Original Issue
*/

#define LGRAPH              0
#define TITLE               1
#define XLABEL              2
#define YLABEL              3
#define XLOGSCALE           4
#define YLOGSCALE           5
#define XGRIDLINES          6
#define YGRIDLINES          7
#define XDIVISIONS          8
#define YDIVISIONS          9
#define RESCALEMODE         10
#define LGRAPHM             11
#define SHUTDOWN            12
#define QUIT                13
#define VIDEOS              14
#define VIDEO               15
#define VQUIT               16
```

```
/*
  SP.C

  Line graph Pop-Up Window Controller.

  Change History:
      12-May-90/mwl - Original Issue.
      23-May-90/mwl - Added support for /dev/graphics & split Lgraph widget.
      02-Aug-90/mwl - Binary transfer Cray to Sun
      25-Sep-90/mwl - Split connection handlers to protocols.c
      03-Nov-90/mwl - Released
*/

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/filio.h>
#include <malloc.h>
#include <string.h>
#include <math.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xol/OpenLook.h>
#include <Xol/ControlAre.h>
#include <Xol/OblongButt.h>
#include <Xol/Form.h>
#include <Xol/BaseWindow.h>
#include <Xol/Caption.h>
#include <Xol/TextField.h>
#include "../Xt+Lgraph/Lgraph.h"
#include <errno.h>
#include "sp.bit"
#include "spicon.bit"
#include "spd.h"
#include "tokens.h"

/* for networking */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SPD_TCP_PORT 10000

extern void quit();
extern void getdata();
extern void connectHandler();

static Arg list[10];

Widget Toplevel;
Pixmap icon_pixmap;
Pixmap spicon_pixmap;

main(argc,argv)
int argc;
char *argv[];
{
  char *lfile;
  char buffer[256];
  char *cptr;
  double x;
  double yv[16];
  int i;
  char tmpbuf[64];
  int infile;
  int insock;
  struct sockaddr_in name;
  int retry;
  COMMCONTEXT *context;
  static char inbuffer[128];
  Widget form,textlabel,quitbutton,loadbutton,filetext,control;
  extern void utilLoad();

  /* initialize toplevel widget and get the environment going */
  Toplevel = OlInitialize("main","LineGraph",NULL,0,&argc,argv);
  icon_pixmap = XCreateBitmapFromData(XtDisplay(Toplevel),XtScreen(Toplevel)-
>root,sp_bits,32,32);
  spicon_pixmap = XCreateBitmapFromData(XtDisplay(Toplevel),XtScreen(Toplevel)-
>root,spicon_bits,64,45);
  XtSetArg(list[0],XtNiconPixmap, spicon_pixmap);
  XtSetValues(Toplevel,list,1);

  /* create the form that controls the main area */
  form = XtCreateManagedWidget("form",formWidgetClass,Toplevel,NULL,0);

  /* create control area for buttons */
  XtSetArg(list[0], XtNxRefWidget, form);
  XtSetArg(list[1], XtNyRefWidget, form);
  XtSetArg(list[2], XtNyResizable,(XtArgVal) FALSE);
  control = XtCreateManagedWidget("control",controlAreaWidgetClass,form,list,3);
```

```
/* create load button */
loadbutton = XtCreateManagedWidget("load",oblongButtonWidgetClass,control,list,0);

/* create a button just to quit if necessary */
quitbutton = XtCreateManagedWidget("quit",oblongButtonWidgetClass,control,list,0);

/* create a text field to get a filename */
XtSetArg(list[0], XtNyAddHeight,(XtArgVal) TRUE);
XtSetArg(list[1], XtNxRefWidget, form);
XtSetArg(list[2], XtNyRefWidget, control);
XtSetArg(list[3], XtNyResizable,(XtArgVal) TRUE);
XtSetArg(list[4], XtNxResizable,(XtArgVal) TRUE);
XtSetArg(list[5], XtNyAttachBottom,(XtArgVal) TRUE);
XtSetArg(list[6], XtNxAttachRight,(XtArgVal) TRUE);
XtSetArg(list[7], XtNxAddWidth,(XtArgVal) TRUE);
textlabel = XtCreateManagedWidget("Filename: ",captionWidgetClass,form,list,8);
XtSetArg(list[0],XtNwidth,300);
filetext = XtCreateManagedWidget("text",textFieldWidgetClass,textlabel,list,1);

/* Add callbacks after everything is defined */
XtAddCallback(loadbutton,XtNselect,utilLoad,filetext);
XtAddCallback(quitbutton,XtNselect,quit,NULL);
XtRealizeWidget(Toplevel); /* let the window system start the display process */

/* create a socket for processes to connect to.
  Use TCP/IP in stream mode.
*/
if ((insock = socket(AF_INET,SOCK_STREAM,0)) < 0)
{
  printf("Error:  Can't open socket\n");
  exit(0);
}
else
{
  if (fcntl(insock,F_SETFL,FNDELAY) == -1)
  {
    fprintf(stderr,"Error:  Can't set FNBIO on connection request socket [%d]\n",errno);
    exit(0);
  }

  bzero((char *)&name,sizeof(name));
  name.sin_family = AF_INET;
  name.sin_port = htons((unsigned short)(SPD_TCP_PORT));
  name.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
    for (retry = 20; bind(insock,&name,sizeof(name)) && retry; retry--)
    {
      fprintf(stderr,"Fatal Error:  Can't bind to address\n");
      exit(0);
    }

    /* listen for connects (Allow up to 5 to queue up) */
    listen(insock,5);

    /* when a connect request comes in, handle it */
    XtAddInput(insock,XtInputReadMask,connectHandler,NULL);
  }


  if ((infile = open("/dev/graphics", O_RDONLY | O_NDELAY)) >= 0)
  {
    /* Add event handlers for /dev/graphics input sources */
    context = (COMMCONTEXT *)malloc(sizeof(COMMCONTEXT));
    context->numgraphs = 0;
    context->firstgraph = NULL;
    context->lastgraph = NULL;
    XtAddInput(infile,XtInputReadMask,getdata,context);
  }
  else
    fprintf(stderr,"Error:  Couldn't open file [/dev/graphics]\n");

  XtMainLoop();
}


void quit()
{
  exit(0);
}
```

```
/*
PROTOCOLS.C

  SP protocol controllers.

  Change History:
      12-May-90/mwl - Original Issue.
      23-May-90/mwl - Added support for /dev/graphics & split Lgraph widget.
      02-Aug-90/mwl - Binary transfer Cray to Sun
      25-Sep-90/mwl - Split off from SP.C and added video protocols
      03-Nov-90/mwl - Released
*/

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/filio.h>
#include <malloc.h>
#include <string.h>
#include <math.h>
#include <X11/Intrinsic.h>
#include "../Xt+Lgraph/Lgraph.h"
#include <errno.h>
#include "spd.h"
#include "tokens.h"

/* for networking */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

extern double atof();
extern char *wordcpy();
extern void getdata();
extern Widget cmdparse();
extern GRAPHCONTEXT *getgraph();
extern void readbinary();
extern void readvideo();
extern void readvideoCompressed();
extern Widget createGraph();
extern Widget createVideo();

static Arg list[10];

static char term[]=" ,\n\t\r";
```

```
void connectHandler(data,connect_handle,id)
caddr_t data;
int *connect_handle;
XtInputId *id;
{
  int insock;
  struct sockaddr addr;
  int addrlen;
  long bytes;
  COMMCONTEXT *context;
  char buffer[128];
  int block;

  if (ioctl(*connect_handle,FIONREAD,&bytes) == -1)
  {
    sprintf(buffer,"Error:  ioctl error on connection handler [%d]",errno);
    utilError(buffer);
    XtRemoveInput(*id);
  }
  else
  {
    if ((insock = accept(*connect_handle,addr,&addrlen)) == -1)
    {
      if (errno == EWOULDBLOCK)  /* if it's a blocking error, just ignore it */
        return;
      sprintf(buffer,"Error:  accept error on connection handler [%d]",errno);
      utilError(buffer);
      XtRemoveInput(*id);
    }

    block = 32766;
    if (setsockopt(insock,SOL_SOCKET,SO_RCVBUF,&block,sizeof(int)) == -1)
    {
      utilError("Error:  can't modify input stream for large buffer size");
    }
    context = (COMMCONTEXT *)malloc(sizeof(COMMCONTEXT));
    context->numgraphs = 0;
    context->firstgraph = NULL;
    context->lastgraph = NULL;
    context->video = NULL;
    XtAddInput(insock,XtInputReadMask,getdata,context);
  }
}
```

```
void getdata(context,handle,id)
COMMCONTEXT *context;
int *handle;
XtInputId *id;
{
  char *cptr;
  double x;
  double yv[16];
  int i;
  long bytes;
  static char buffer[1024];
  static char tmpbuf[1024];
  Widget graph;

  /* Make sure that this is still a valid socket connection */
  if (ioctl(*handle,FIONREAD,&bytes) == -1)
  {
    XtRemoveInput(*id);
    close(*handle);
    utilError("Error:  Invalid Connection");
    return;
  }

  /* if there is nothing left to read, but we got called to read data,
    something is wrong, so close this connection. */
  if (!bytes)
  {
    XtRemoveInput(*id);
    close(*handle);
    utilError(stderr,"Error:  Connection Abruptly Terminated");
    return;
  }

  for (cptr = buffer; ; cptr++)
  {
    mread(*handle,cptr,1);

    /* if the first character is a !, @ or a $, the packet is binary
      graph or video data.
    */
    if (cptr == buffer)
    {
      switch (*cptr)
      {
        case '!':
          readbinary(*handle,context);
          return;

        case '@':
          readvideo(*handle,context);
          return;

        case '$':
          readvideoCompressed(*handle,context);
          return;
      }
    }
    /* if this is a newline character, it is the end of this text record */
    if (*cptr == '\n')
    {
      *cptr = '\0';
      break;
    }
  }

  if (buffer[0] == '#')  /* ASCII Text Protocol */
    cmdparse(buffer,context,*id,*handle);
  else  /* no protocol */
  {
    cptr = wordcpy(tmpbuf,buffer,term);
    x = atof(tmpbuf);

    for (i = 0; (cptr = wordcpy(tmpbuf,cptr,term)) != NULL; i++)
      yv[i] = atof(tmpbuf);

    if (context->firstgraph && i) /* if a graph exists, write to it */
    {
      lgraphAddData(context->firstgraph->graph,x,yv,i);
    }
  }
}


/*
  readvideo()

  Process a binary color video data request.
*/
```

```
static void readvideo(handle,context)
int handle;
COMMCONTEXT *context;
{
  if (!context->video)
  {
    utilError("Error:  No video connection established");
    return;
  }

  mread(handle,context->video->buffer,context->video->buflength);

  if (context->video->buffer[context->video->buflength - 1] != '\n')
    utilError("Invalid video block");
  else
    ImageScientificSet(context->video->video,context->video->buffer);
}

/*
  readvideoCompressed()

  Process a binary color video data request.
*/

static void readvideoCompressed(handle,context)
int handle;
COMMCONTEXT *context;
{
  long length[2];

  if (!context->video)
  {
    utilError("Error:  No video connection established");
    return;
  }

  mread(handle,length,8);
  /* read in the bytes contained by the delta buffer + 1 newline */
#ifdef SPDEBUG
fprintf(stderr,"[%d]",length[1]);
#endif
  mread(handle,context->video->buffer,length[1] + 1);

  ImageDeltaSet(context->video->video,context->video->buffer);
}
```

```
/*
  readbinary()

  Process a binary graph data request.
*/
static void readbinary(handle,context)
int handle;
COMMCONTEXT *context;
{
  unsigned char buffer[512];
  int yvc;
  int graph;
  GRAPHCONTEXT *graphcontext;
  double fb[32];
  int i;

  mread(handle,buffer,7);

  graph = (unsigned int)buffer[0];
  yvc = (unsigned int)buffer[1];

#ifdef SPDEBUG
printf("\nGraph: %d  YVC: %d - ",graph,yvc);
#endif

  mread(handle,fb,8 + 8 * yvc);

#ifdef SPDEBUG
for (i = 0; i < yvc; i++)
  printf("%lx ",fb[i]);
#endif


  mread(handle,buffer,1);

  if (buffer[0] != '\n')
  {
    utilError("Error:  Unbound binary block - Data was lost");
    return;
  }

  if (!(graphcontext = getgraph(graph,context)))
  {
    utilError("Error:  Bad graph handle");
    return;
  }

  lgraphAddData(graphcontext->graph,fb[0],&fb[1],yvc);
}

static int mread(handle,buffer,length)
int handle;
char *buffer;
int length;
{
  int retry = 15;
  char tmpbuf[64];
  int bytes;

#ifdef SPDEBUG
fprintf(stderr,"Attempting to read [%d] bytes\n",length);
#endif

  while (retry--)
  {
    bytes = read(handle,buffer,length);
    if (bytes == -1)  /* error, no bytes read */
    {
      /* sometimes operation can block the process, this is OK */
      if (errno != EWOULDBLOCK && errno != EINTR)
      {
        sprintf(tmpbuf,"Error:  TCP/IP Channel Read Error [%d]",errno);
        utilError(tmpbuf);
      }
      else
      {
        sleep(1);
        retry++;
#ifdef SPDEBUG
fprintf(stderr,".");
#endif
      }
    }
    else if (bytes == length)  /* got exactly what I want */
      return (bytes);
    else /* read insufficient data, try again for the rest */
    {
      length -= bytes;
      buffer += bytes;
#ifdef SPDEBUG
fprintf(stderr,"+");
#endif
      retry++;
    }
  }
}

static Widget cmdparse(buffer,context,id,handle)
char *buffer;
COMMCONTEXT *context;
XtInputId id;
int handle;
{
  char *cptr;
  char tmpbuf[128];
  int width,height,numcolors;
  char lowcolor[64],highcolor[64];
  float minX,minY,maxX,maxY;
  char xlabel[64],ylabel[64],title[64];
  Widget graph;
  int graphno;
  int i;
  double x;
  double yv[16];
  GRAPHCONTEXT *graphcontext;
  int value;

  cptr = wordcpy(tmpbuf,buffer,term);
  switch (command(&tmpbuf[1]))
  {
    case VIDEOS:  /* open for spectrum video connection */
      cptr = wordcpy(tmpbuf,cptr,term);
      width = atoi(tmpbuf);
      cptr = wordcpy(tmpbuf,cptr,term);
      height = atoi(tmpbuf);
      cptr = wordcpy(lowcolor,cptr,term);
      cptr = wordcpy(highcolor,cptr,term);
      cptr = wordcpy(tmpbuf,cptr,term);
      numcolors = atoi(tmpbuf);
      if (!context->video)
        createVideo(context,width,height,lowcolor,highcolor,numcolors);
      else
        utilError("Error:  Can only have one open video connection");
      break;
    case VIDEO:  /* open for discrete colormap video connection */
      break;

    case VQUIT:  /* close the video connection */
      free(context->video->buffer);  /* free the I/O memory */
      context->video->buffer = NULL;
      context->video->buflength = 0;
      context->video->connected = 0;
      context->video = NULL;  /* turn it loose */
      break;

   case LGRAPHM: /* open for multiple graph connections, make sure to send the graphno
back */
      cptr = wordcpy(tmpbuf,cptr,term);
      minX = atof(tmpbuf);
      cptr = wordcpy(tmpbuf,cptr,term);
      minY = atof(tmpbuf);
      cptr = wordcpy(tmpbuf,cptr,term);
      maxX = atof(tmpbuf);
      cptr = wordcpy(tmpbuf,cptr,term);
      maxY = atof(tmpbuf);
      if (cptr)  /* optional label arguments */
      {
        cptr = wordcpy(xlabel,cptr,term);
        cptr = wordcpy(ylabel,cptr,term);
        cptr = wordcpy(title,cptr,term);
        graph = createGraph(minX,minY,maxX,maxY,xlabel,ylabel,title);
      }
      else
        graph = createGraph(minX,minY,maxX,maxY,NULL,NULL,NULL);

      graphno = addgraph(graph,context);
      sprintf(tmpbuf,"%d\n",graphno);
     write(handle,tmpbuf,strlen(tmpbuf));  /* send back confirmation of the handle number
*/
      break;

    case LGRAPH:  /* open for just a single graph (don't wait for confirmation) */
      cptr = wordcpy(tmpbuf,cptr,term);
      minX = atof(tmpbuf);
      cptr = wordcpy(tmpbuf,cptr,term);
      minY = atof(tmpbuf);
      cptr = wordcpy(tmpbuf,cptr,term);
      maxX = atof(tmpbuf);
```

```
  cptr = wordcpy(tmpbuf,cptr,term);
  maxY = atof(tmpbuf);
  if (cptr)  /* optional label arguments */
  {
    cptr = wordcpy(xlabel,cptr,term);
    cptr = wordcpy(ylabel,cptr,term);
    cptr = wordcpy(title,cptr,term);
    graph = createGraph(minX,minY,maxX,maxY,xlabel,ylabel,title);
  }
  else
    graph = createGraph(minX,minY,maxX,maxY,NULL,NULL,NULL);

  graphno = addgraph(graph,context);
  break;


case TITLE:   /* #title value [graphno] */
  break;

case XLABEL:  /* #xlabel value [graphno] */
  break;

case YLABEL:  /* #ylabel value [graphno] */
  break;

case XLOGSCALE:  /* #xlogscale value [graphno] */
  break;

case YLOGSCALE:  /* #ylogscale value [graphno] */
  break;

case XDIVISIONS: /* #xdivisions value [graphno] */
  cptr = wordcpy(tmpbuf,cptr,term);
  value = atoi(tmpbuf);

  if (cptr)  /* if a graphno is there, get it, otherwise default */
  {
    cptr = wordcpy(tmpbuf,cptr,term);
    graphno = atoi(tmpbuf);
    if (!(graphcontext = getgraph(graphno,context)))
    {
      utilError("Error:  Bad graph handle");
      break;  /* this is bad data, throw it away */
    }
  }
```

```
    else /* default graph is first in the list */
      graphcontext = context->firstgraph;;

    XtSetArg(list[0],XtNxdivisions,value);
    XtSetValues(graphcontext->graph,list,1);
    break;

  case YDIVISIONS: /* #ydivisions value [graphno] */
    cptr = wordcpy(tmpbuf,cptr,term);
    value = atoi(tmpbuf);

    if (cptr)  /* if a graphno is there, get it, otherwise default */
    {
      cptr = wordcpy(tmpbuf,cptr,term);
      graphno = atoi(tmpbuf);
      if (!(graphcontext = getgraph(graphno,context)))
      {
        utilError("Error:  Bad graph handle");
        break;  /* this is bad data, throw it away */
      }
    }
    else /* default graph is first in the list */
      graphcontext = context->firstgraph;;

    XtSetArg(list[0],XtNydivisions,value);
    XtSetValues(graphcontext->graph,list,1);
    break;

  case XGRIDLINES: /* #xgridlines value [graphno] */
    cptr = wordcpy(tmpbuf,cptr,term);
    value = (tmpbuf[0] == 'T') ? 1 : 0;

    if (cptr)  /* if a graphno is there, get it, otherwise default */
    {
      cptr = wordcpy(tmpbuf,cptr,term);
      graphno = atoi(tmpbuf);
      if (!(graphcontext = getgraph(graphno,context)))
      {
        utilError("Error:  Bad graph handle");
        break;  /* this is bad data, throw it away */
      }
    }
    else /* default graph is first in the list */
      graphcontext = context->firstgraph;;
    XtSetArg(list[0],XtNxdividers,value);
    XtSetValues(graphcontext->graph,list,1);
    break;

  case YGRIDLINES: /* #ygridlines value [graphno] */
    cptr = wordcpy(tmpbuf,cptr,term);
    value = (tmpbuf[0] == 'T') ? 1 : 0;

    if (cptr)  /* if a graphno is there, get it, otherwise default */
    {
      cptr = wordcpy(tmpbuf,cptr,term);
      graphno = atoi(tmpbuf);
      if (!(graphcontext = getgraph(graphno,context)))
      {
        utilError("Error:  Bad graph handle");
        break;  /* this is bad data, throw it away */
      }
    }
    else  /* default graph is first in the list */
      graphcontext = context->firstgraph;;

    XtSetArg(list[0],XtNydividers,value);
    XtSetValues(graphcontext->graph,list,1);
    break;

  case QUIT:      /* #quit [graphno] */
    if (cptr)
    {
      cptr = wordcpy(tmpbuf,cptr,term);
      graphno = atoi(tmpbuf);
    }
    else if (context->firstgraph)
    {
      if (context->firstgraph == context->lastgraph)  /* is there only one? */
        context->lastgraph = NULL;
      context->firstgraph = context->firstgraph->nextgraph;
      free(context->firstgraph);
      break;
    }
    removegraph(graphno,context);
break;

  case SHUTDOWN:  /* #shutdown */
    for (graphcontext = context->firstgraph; graphcontext; graphcontext = graphcontext-
>nextgraph)
```

```
        free(graphcontext); /* get rid of the communication contexts, but leave the graphs to
display */

      if (context->video)  /* if there is a video connection */
        context->video->connected = 0;  /* release it */

      free(context);
      XtRemoveInput(id);
      close(handle);
      break;

    default:      /* #graphno x yv[0] yv[1] yv[2] ....... */
      graphno = atoi(&tmpbuf[1]);
      if (!(graphcontext = getgraph(graphno,context)))
      {
        utilError("Error:  Bad graph handle");
        break;  /* this is bad data, throw it away */
      }

      cptr = wordcpy(tmpbuf,cptr,term);
      x = atof(tmpbuf);

      for (i = 0; (cptr = wordcpy(tmpbuf,cptr,term)) != NULL; i++)
        yv[i] = atof(tmpbuf);

      lgraphAddData(graphcontext->graph,x,yv,i);
      break;
  }
}

GRAPHCONTEXT *getgraph(graphno,context)
int graphno;
COMMCONTEXT *context;
{
  GRAPHCONTEXT *graphcontext;

  for (graphcontext = context->firstgraph; graphcontext; graphcontext = graphcontext->nex-
tgraph)
    if (graphcontext->graphno == graphno)
      return (graphcontext);

  return (NULL);
}

int addgraph(graph,context)
```

```
Widget graph;
COMMCONTEXT *context;
{
  GRAPHCONTEXT *graphcontext;

  graphcontext = (GRAPHCONTEXT *)malloc(sizeof(GRAPHCONTEXT));
  if (!context->lastgraph)
    context->firstgraph = graphcontext;
  else
    context->lastgraph->nextgraph = graphcontext;

  context->lastgraph = graphcontext;
  graphcontext->nextgraph = NULL;
  graphcontext->graph = graph;
  graphcontext->graphno = context->numgraphs;
  context->numgraphs++;  /* one more graph */

  return (graphcontext->graphno);
}

removegraph(graphno,context)
int graphno;
COMMCONTEXT *context;
{
  GRAPHCONTEXT *lastgraph;
  GRAPHCONTEXT *graphcontext;

  lastgraph = NULL;
  for (graphcontext = context->firstgraph; graphcontext; graphcontext = graphcontext->nex-
tgraph)
  {
    if (graphcontext->graphno == graphno)
    {
      if (lastgraph)
        lastgraph->nextgraph = graphcontext->nextgraph;
      else
        context->firstgraph = graphcontext->nextgraph;

      if (graphcontext == context->lastgraph)
        context->lastgraph = lastgraph;

      free(graphcontext);
      return (1);
    }
    lastgraph = graphcontext;
  }
  return (0);
}

static char *wordcpy(dst,src,terminators)
char *dst;
char *src;
char *terminators;
{
  if (!src || !*src)
  {
   *dst = 0x0;
   return (NULL);
  }

  while (bchr(terminators,*src))
    src++;

  if (!*src)
  {
    *dst = 0x0;
    return (NULL);
  }

  if (*src == '\"')
  {
    src++;
    while (*src != '\"' && *src)
    {
      *dst = *src;
      dst++;
      src++;
    }
    if (*src == '\"')
      src++;
  }
  else
  {
    while (!bchr(terminators,*src) && *src)
    {
      *dst = *src;
      dst++;
      src++;
    }
  }
  *dst = 0x0;

  return (src);
}

static int bchr(buf,ch)
char *buf;
char ch;
{
  while (*buf)
  {
    if (*buf == ch)
      return (1);
    buf++;
  }
  return (0);
}
```

```
/*
  TOKEN.C

  Change History:
      9-Jul-90/mwl - Original Issue
      3-Nov-90/mwl - Released
*/

#include <string.h>
#include "tokens.h"

static char *Tokenlist[] =
{
  "LGRAPH",
  "TITLE",
  "XLABEL",
  "YLABEL",
  "XLOGSCALE",
  "YLOGSCALE",
  "XGRIDLINES",
  "YGRIDLINES",
  "XDIVISIONS",
  "YDIVISIONS",
  "RESCALEMODE",
  "LGRAPHM",
  "SHUTDOWN",
  "QUIT",
  "VIDEOS",
  "VIDEO",
  "VQUIT",
  NULL
};

int command(cmdbuf)
char *cmdbuf;
{
  int i;

  for (i = 0; Tokenlist[i]; i++)
    if (!strcasecmp(cmdbuf,Tokenlist[i]))
      return (i);

  return (-1);  /* didn't find anything */
}
```

```
/*
  UTILITIES.C

  Change History:
     6-Jul-90/mwl - Original Issue
     3-Nov-90/mwl - Released
*/
#include <stdio.h>
#include <fcntl.h>
#include <malloc.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xol/OpenLook.h>
#include <Xol/BaseWindow.h>
#include <Xol/ControlAre.h>
#include <Xol/Form.h>
#include <Xol/ControlAre.h>
#include <Xol/OblongButt.h>
#include <Xol/Caption.h>
#include <Xol/TextField.h>
#include <Xol/PopupWindo.h>
#include <Xol/BaseWindow.h>
#include <Xol/Notice.h>
#include <sys/param.h>
#include <X11/IntrinsicP.h>
#include "../Xt+Lgraph/LgraphP.h"
#include "../Image/Image.h"
#include "spd.h"

extern void destroyVideo();
extern void destroyGraph();
extern void exportGraph();
extern void utilSave();
extern void utilError();

extern Widget Toplevel;
extern Pixmap icon_pixmap;

static char Filename[128] = "";

static void destroyGraph(w,graphApp,event)
Widget w;
GRAPHAPP *graphApp;
caddr_t event;
{
  XtPopdown(graphApp->shell);
  XtDestroyWidget(graphApp->shell);
  free((caddr_t)graphApp);
}

static void exportGraph(w,graphApp,event)
Widget w;
GRAPHAPP *graphApp;
caddr_t event;
{
  FILE *fp;
  char buffer[128];
  LgraphWidget graph;
  LDATA *ldata;
  int i;
  Arg list[1];
  char *filename;
  int datapts;

  XtSetArg(list[0],XtNstring,&filename);
  XtGetValues(graphApp->text,list,1);

  if (!filename || !strlen(filename))
  {
    utilError("Error:  No file specified.");
    return;
  }

  strcpy(Filename,filename);  /* save the filename so it can be used later */

  fp = fopen(filename,"w");
  if (!fp)
  {
    sprintf(buffer,"Error:  Couldn't open file [%s] for write",filename);
    utilError(buffer);
    return;
  }

  graph = (LgraphWidget)graphApp->graph;

  fprintf(fp,"Device: Postscript\nDisposition: To Device\nFileOrDev: lp\n");
  fprintf(fp,"TitleText: %s\n",graph->lgraph.title);

  datapts = graph->lgraph.ldata->yvc;
  for (i = 0; i < datapts; i++)
  {
    fprintf(fp,"\n\"%d\n",i);
    for (ldata = graph->lgraph.ldata; ldata; ldata = ldata->nextdata)
    {
      sprintf(buffer,"%g %g\n",ldata->x,ldata->yv[i]);
      fputs(buffer,fp);
    }
  }

  fclose(fp);
}

static void utilSave(w,graphApp,event)
Widget w;
GRAPHAPP *graphApp;
caddr_t event;
{
  FILE *fp;
  char buffer[128];
  LgraphWidget graph;
  LDATA *ldata;
  int i;
  Arg list[1];
  char *filename;

  XtSetArg(list[0],XtNstring,&filename);
  XtGetValues(graphApp->text,list,1);

  if (!filename || !strlen(filename))
  {
    utilError("Error:  No file specified.");
    return;
  }

  strcpy(Filename,filename);  /* save the filename so it can be used later */

  fp = fopen(filename,"w");
  if (!fp)
  {
    sprintf(buffer,"Error:  Couldn't open file [%s] for write",filename);
    utilError(buffer);
    return;
  }

  graph = (LgraphWidget)graphApp->graph;
  fprintf(fp,"#lgraph %g %g %g %g \"%s\" \"%s\" \"%s\" \n",
graph->lgraph.minX,
graph->lgraph.minY,
graph->lgraph.maxX,
graph->lgraph.maxY,
graph->lgraph.xlabel,
graph->lgraph.ylabel,
graph->lgraph.title);

  fprintf(fp,"#xgridlines %s\n",(graph->lgraph.xdividers) ? "TRUE" : "FALSE");
  fprintf(fp,"#ygridlines %s\n",(graph->lgraph.ydividers) ? "TRUE" : "FALSE");
  fprintf(fp,"#xdivisions %d\n",graph->lgraph.xdivisions);
  fprintf(fp,"#ydivisions %d\n",graph->lgraph.ydivisions);

  for (ldata = graph->lgraph.ldata; ldata; ldata = ldata->nextdata)
  {
    sprintf(buffer,"%g",ldata->x);
    for (i = 0; i < ldata->yvc; i++)
      sprintf(&buffer[strlen(buffer)]," %g",ldata->yv[i]);
    strcat(buffer,"\n");
    fputs(buffer,fp);
  }

  fputs("#shutdown\n",fp);
  fclose(fp);
}

extern void getdata();

void utilLoad(w,filetext,event)
Widget w;
Widget filetext;
caddr_t event;
{
  int infile;
  COMMCONTEXT *context;
  char buffer[128];
  char *filename;
  Arg list[2];

  XtSetArg(list[0],XtNstring,&filename);
  XtGetValues(filetext,list,1);

  if (!filename || !strlen(filename))
  {
    utilError("Error:  No file specified.");
    return;
  }

  if ((infile = open(filename, O_RDONLY | O_NDELAY)) >= 0)
  {
    /* Add event handlers for /dev/graphics input sources */
    context = (COMMCONTEXT *)malloc(sizeof(COMMCONTEXT));
    context->numgraphs = 0;
    context->firstgraph = NULL;
    context->lastgraph = NULL;
    XtAddInput(infile,XtInputReadMask,getdata,context);
  }
  else
  {
    sprintf(buffer,"Error:  Couldn't open file [%s]\n",filename);
    utilError(buffer);
  }
}

void createVideo(context,width,height,lowcolor,highcolor,numcolors)
COMMCONTEXT *context;
int width;
int height;
char *lowcolor;
char *highcolor;
int numcolors;
{
  Widget control,video,shell,form,destroy;
  Arg list[10];
  VIDEOCONTEXT *videocontext;

#ifdef SPDEBUG
fprintf(stderr,"Creating Video: %d %d %s %s %d\n",width,height,lowcolor,highcol-
or,numcolors);
#endif

  if (context->video)
  {
    fprintf(stderr,"Error:  video connection already present\n");
    utilError("Error:  video connection already present");
    return;
  }
```

```
    XtSetArg(list[0],XtNtitle, "SPD Video");
    XtSetArg(list[1],XtNiconName, "SPD Video");
    XtSetArg(list[2],XtNiconPixmap, icon_pixmap);
    shell = XtCreatePopupShell("spd",baseWindowShellWidgetClass,Toplevel,list,3);

    form = XtCreateManagedWidget("form",formWidgetClass,shell,list,0);

    XtSetArg(list[0], XtNxRefWidget, form);
    XtSetArg(list[1], XtNyRefWidget, form);
    XtSetArg(list[2], XtNyResizable,(XtArgVal) FALSE);
    control = XtCreateManagedWidget("control",controlAreaWidgetClass,form,list,3);

    destroy = XtCreateManagedWidget("destroy",oblongButtonWidgetClass,control,list,0);

    XtSetArg(list[0],XtNwidth,width);
    XtSetArg(list[1],XtNheight,height);
    XtSetArg(list[2],XtNlowcolor,lowcolor);
    XtSetArg(list[3],XtNhighcolor,highcolor);
    XtSetArg(list[4],XtNnumcolors,numcolors);
    XtSetArg(list[5], XtNyAddHeight,(XtArgVal) TRUE);
    XtSetArg(list[6], XtNxRefWidget, form);
    XtSetArg(list[7], XtNyRefWidget, control);
    XtSetArg(list[8], XtNxAddWidth,(XtArgVal) TRUE);
    video = XtCreateManagedWidget("Video",imageWidgetClass,form,list,9);

    videocontext = (VIDEOCONTEXT *)malloc(sizeof(VIDEOCONTEXT));
    videocontext->video = video;
    videocontext->buffer = malloc(width * height + 1);
    videocontext->buflength = width * height + 1;
    videocontext->videoShell = shell;
    videocontext->connected = 1;  /* do not destroy until disconnected */
    context->video = videocontext;

    XtAddCallback(destroy,XtNselect,destroyVideo,(caddr_t)videocontext);

    XtPopup(shell,XtGrabNone);
}

static void destroyVideo(w,context,event)
Widget w;
VIDEOCONTEXT *context;
caddr_t event;
{
  if (context->connected)
  {
    utilError("Error: Cannot destroy graph until it has disconnected");
    return;
  }

  if (context->buffer)
    free(context->buffer);

  context->buflength = 0;
  if (context->videoShell);
  {
    XtPopdown(context->videoShell);
    XtDestroyWidget(context->videoShell);
  }

  free((char *)context);
}

Widget createGraph(minX,minY,maxX,maxY,xlabel,ylabel,title)
float minX,minY,maxX,maxY;
char *xlabel, *ylabel, *title;
{
  Widget destroy,export,save,shell,graph,form,control,caption,text;
  char *ixlabel,*iylabel,*ititle;
  GRAPHAPP *appdata;
  Arg list[15];

  XtSetArg(list[0],XtNtitle,(title) ? title : "SPD");
  XtSetArg(list[1],XtNiconName,(title) ? title : "SPD");
  XtSetArg(list[2],XtNiconPixmap, icon_pixmap);
  shell = XtCreatePopupShell("spd",baseWindowShellWidgetClass,Toplevel,list,3);

  form = XtCreateManagedWidget("form",formWidgetClass,shell,list,0);

  XtSetArg(list[0], XtNxRefWidget, form);
  XtSetArg(list[1], XtNyRefWidget, form);
  XtSetArg(list[2], XtNyResizable,(XtArgVal) FALSE);
  control = XtCreateManagedWidget("control",controlAreaWidgetClass,form,list,3);

  destroy = XtCreateManagedWidget("destroy",oblongButtonWidgetClass,control,list,0);

  export = XtCreateManagedWidget("export",oblongButtonWidgetClass,control,list,0);

  save = XtCreateManagedWidget("save",oblongButtonWidgetClass,control,list,0);

  caption = XtCreateManagedWidget("Filename: ",captionWidgetClass,control,list,0);
```

```
 XtSetArg(list[0], XtNwidth,300);
 text = XtCreateManagedWidget("textfield",textFieldWidgetClass,caption,list,1);

 if (xlabel && ylabel)
 {
   ixlabel = malloc(strlen(xlabel) + 1);
   strcpy(ixlabel,xlabel);
   iylabel = malloc(strlen(ylabel) + 1);
   strcpy(iylabel,ylabel);
   XtSetArg(list[0],XtNxlabel,ixlabel);
   XtSetArg(list[1],XtNylabel,iylabel);
   XtSetArg(list[2], XtNyAddHeight,(XtArgVal) TRUE);
   XtSetArg(list[3], XtNxRefWidget, form);
   XtSetArg(list[4], XtNyRefWidget, control);
   XtSetArg(list[5], XtNyResizable,(XtArgVal) TRUE);
   XtSetArg(list[6], XtNxResizable,(XtArgVal) TRUE);
   XtSetArg(list[7], XtNyAttachBottom,(XtArgVal) TRUE);
   XtSetArg(list[8], XtNxAttachRight,(XtArgVal) TRUE);
   XtSetArg(list[9], XtNxAddWidth,(XtArgVal) TRUE);
   ititle = malloc(strlen(title) + 1);
   strcpy(ititle,title);
   XtSetArg(list[10],XtNtitle,title);
   graph = XtCreateManagedWidget("Graph",lgraphWidgetClass,form,list,11);
 }
 else
 {
   XtSetArg(list[0], XtNyAddHeight,(XtArgVal) TRUE);
   XtSetArg(list[1], XtNxRefWidget, form);
   XtSetArg(list[2], XtNyRefWidget, control);
   XtSetArg(list[3], XtNyResizable,(XtArgVal) TRUE);
   XtSetArg(list[4], XtNxResizable,(XtArgVal) TRUE);
   XtSetArg(list[5], XtNyAttachBottom,(XtArgVal) TRUE);
   XtSetArg(list[6], XtNxAttachRight,(XtArgVal) TRUE);
   XtSetArg(list[7], XtNxAddWidth,(XtArgVal) TRUE);
   XtSetArg(list[8], XtNtitle,title);
   graph = XtCreateManagedWidget("Graph",lgraphWidgetClass,form,list,9);
 }

lgraphSetLimits(graph,minX,minY,maxX,maxY,10); /* let's try to get a smooth display */

/* add these hooks so that the utilities can get to these widgets
  and work on them.
*/
appdata = (GRAPHAPP *)malloc(sizeof(GRAPHAPP));
appdata->shell = shell;
```

```
  appdata->graph = graph;
  appdata->text = text;
  appdata->connected = 1;  /* currently connected to a file */

  XtAddCallback(destroy,XtNselect,destroyGraph,(caddr_t)appdata);
  XtAddCallback(export,XtNselect,exportGraph,(caddr_t)appdata);
  XtAddCallback(save,XtNselect,utilSave,(caddr_t)appdata);

  XtPopup(shell,XtGrabNone);

  return (graph);
}

/*
  ERROR HANDLING SUBROUTINES

*/
static Widget errshell = NULL;
static char *ibuf;

void utilErrorOK(w,inbuffer,event)
Widget w;
char *inbuffer;
caddr_t event;
{
  XtPopdown(errshell);
  XtDestroyWidget(errshell);
  errshell = NULL;
  free(ibuf);
}

void utilError(buffer)
char *buffer;
{
  Arg list[5];
  Widget ok,text,control;

  if (errshell)
    return;

fprintf(stderr,buffer);
  /* allocate my own internal buffer for the message */
  ibuf = malloc(strlen(buffer) + 1);
  strcpy(ibuf,buffer);
  XtSetArg(list[0],XtNstring,ibuf);
  XtSetArg(list[1],XtNtitle,"SPD");
  errshell = XtCreatePopupShell("notice",noticeShellWidgetClass,Toplevel,list,2);

  XtSetArg(list[0],XtNcontrolArea, (XtArgVal)&control),
  XtGetValues(errshell,list,1);

  ok = XtCreateManagedWidget("OK",oblongButtonWidgetClass,control,list,0);
  XtAddCallback(ok,XtNselect,utilErrorOK,NULL);

  XtPopup(errshell,XtGrabExclusive);
}
```

APPENDIX H

LIBSP SOURCES

Source Code Listings for Libsp

```
/*

    SP.H

    Header file for network display interface SP.

    Change History:
      28-Jun-90/mwl - Original Issue
      16-Sep-90/mwl - Added TRUE, FALSE, and IGNORE directives
      26-Sep-90/mwl - Added Image directives
*/

struct ImageRec
{
  int buflength;
  char *data;
  int displayValid;
};

typedef int Graph;
typedef int Display;
typedef struct ImageRec *Image;

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

#define IGNORE -1
#define BLANK NULL

extern Image SpImageOpen();
```

```
/*
  libsp.c

  Network Interface for SP graphics.

  Change History:
      18-Jun-90/mwl - Original Issue
      27-Jun-90/mwl - Reworked into Sp library format
      03-Nov-90/mwl - Released
*/
#include <stdio.h>
#include <malloc.h>
#include <ctype.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <sys/errno.h>
#include <netdb.h>
#include "sp.h"

#define SPD_TCP_PORT 10000
#define DELTABUFSIZE 128000

extern char *getenv();

#ifdef CRAY
extern unsigned int crayFloatToIEEE();
#endif

Display SpInitialize(argc,argv)
int *argc;
char *argv[];
{
  int handle;
  int addrlen;
  struct sockaddr_in name;
  char *hostid = "192.58.110.14";
  unsigned long hostaddr;
  struct hostent *host;
  unsigned long *hostptr;
  int i,j;
  char *cptr;
  char *envptr;
  char envbuffer[128];

  /* set the default host address */
  hostaddr = inet_addr(hostid);  /* by default */

  /* if the DISPLAY environment variable is used, use that value by default */
  if ((envptr = getenv("DISPLAY")))
  {
    strcpy(envbuffer,envptr);
    if ((cptr = strchr(envbuffer,':')))
      *cptr = '\0';  /* if they used an X display string, truncate */
    if (isdigit(envbuffer[0]))
    {
      if ((hostaddr = inet_addr(envbuffer)) == -1)
        fprintf(stderr,"Warning:  Can't resolve TCP/IP Address in DISPLAY variable
[%s]\n",envbuffer);
    }
    else
    {
      host = gethostbyname(envbuffer);
      if (host)
      {
#ifdef CRAY
        hostaddr = *(unsigned long *)host->h_addr_list[0] >> 32;
#else
        hostaddr = *(unsigned long *)host->h_addr_list[0];
#endif
      }
      else
        fprintf(stderr,"Warning: Can't resolve hostname in DISPLAY variable [%s]\n",en-
vbuffer);
    }
  }

  /* check each individual display option */
  for (i = 1; i < *argc; i ++)
  {
    if (argv[i][0] == '-')
    {
      switch (argv[i][1])
      {
        case 'd':  /* -display hostname */
          if ((cptr = strchr(argv[i + 1],':')))
            *cptr = '\0';  /* if they used an X display string, truncate */
          if (isdigit(argv[i + 1][0]))
          {
            if ((hostaddr = inet_addr(argv[i + 1])) == -1)
              fprintf(stderr,"Warning:  Can't resolve TCP/IP Address [%s]\n",argv[i+1]);
          }
          else
          {
            host = gethostbyname(argv[i + 1]);
            if (host)
            {
#ifdef CRAY
              hostaddr = *(unsigned long *)host->h_addr_list[0] >> 32;
#else
              hostaddr = *(unsigned long *)host->h_addr_list[0];
#endif
#ifdef DEBUG
              fprintf(stderr,"Real Name: %s   Address: %lx\n",host->h_name,hostaddr);
#endif
            }
            else
            {
              fprintf(stderr,"Warning:  Can't resolve hostname [%s]\n",argv[i+1]);
            }
          }

          /* get rid of these options */
          for (j = i + 2; j <= *argc; j++)
            argv[j - 2] = argv[j];
          *argc -= 2;
          break;

        default:
          break;
      }
    }
  }

  /* create a socket for processes to connect to.
     Use TCP/IP in stream mode.
  */
  handle = socket(PF_INET,SOCK_STREAM,0);

  name.sin_family = AF_INET;
  name.sin_port = SPD_TCP_PORT;
  name.sin_port = htons (name.sin_port);
  name.sin_addr.s_addr = hostaddr;

  if (connect(handle,&name,sizeof(name)))
  {
    printf("Error:  Can't connect to spd daemon[%d]\n",errno);
    return (-1);
  }

  return (handle);
}

/
**********************************************************************
*****************
*  Image routines                                        *
*                                                    *
*                                                    *
**********************************************************************
*****************/

Image SpImageOpen(display,width,height,lowcolor,highcolor,numcolors)
Display display;
int width,height;
char *lowcolor;
char *highcolor;
int numcolors;
{
  char buffer[128];
  Image image;

  sprintf(buffer,"#videos %d %d \"%s\" \"%s\" %d \n",width,height,lowcolor,highcolor,-
numcolors);
  write(display,buffer,strlen(buffer));

  image = (Image)malloc(sizeof(Image));
  image->data = malloc(width * height);
  image->buflength = width * height;
  image->displayValid = FALSE;

  return (image);
}

int SpImageClose(display,image)
Display display;
Image image;
```

```
{
  write(display,"#vquit\n",7);
  free(image);
  return(0);
}

int SpDumpDelta(buffer)
unsigned char *buffer;
{
  int i;

  while (*buffer != 'D')
  {
    switch (*buffer)
    {
      case 'U':
        printf("U: [%u] ",*(buffer + 1));
        for (i = 0; i < *(buffer + 1); i++)
          printf("%02x ",*(buffer + 2 + i));
        printf("\n");
        buffer += *(buffer + 1) + 2;
        break;

      case 'S':
        printf("S: [%u]\n",*(buffer + 1));
        buffer += 2;
        break;

     default:
        printf("Error:  Invalid type [%c]\n",*buffer);
        buffer++;
        break;
    }
  }
}

static unsigned char deltabuffer[DELTABUFSIZE];

int SpImageGenDelta(old,new,size,delta,maxdelta)
unsigned char old[];
unsigned char new[];
unsigned long size;
unsigned char *delta;
unsigned long maxdelta;
{
```

```
int bytes,i;
unsigned long position = 0;
unsigned char *length;
unsigned char *cptr;
unsigned int cmdpos;
unsigned long indelta = 0;

static unsigned char command[258];
static int bailout = 0;
static int saved_position;
static int saved_cmdpos;

/* if the last delta calculation failed due to lack of delta buffer space,
  continue where we left off.
*/
if (bailout)
{
  bailout = 0;
  position = saved_position;
  for (i = 0; i < saved_cmdpos; i++)
    *delta++ = command[i];

  indelta += saved_cmdpos;
}

/* start calculating delta data.
*/
while (!bailout && indelta < maxdelta  - 1 && position < size)
{
  command [0] = 'D';
  cmdpos = 1;
  bytes = 0;

  while (old[position] == new[position] && bytes < 255 && position < size)
  {
    position++;
    bytes++;
  }

  if (bytes < 5 && position < size)
  {
    command[0] = 'U';
    length = &command[1];
    *length = 0;
    cmdpos = 2;
```

```
      for (i = position - bytes; i < position; i++)
      {
        command[cmdpos++] = new[i];
        (*length)++;
      }

      while (old[position] != new[position] && *length < 255 && position < size)
      {
        command[cmdpos++] = new[position];
        position++;
        (*length)++;
      }

      if (*length == 0)
      {
        command[0] = 'D';
        cmdpos = 1;
      }
    }
    else if (position < size)
    {
      command[0] = 'S';
      command[1] = bytes;
      cmdpos = 2;
    }

    /* add the command to the delta buffer */
    if (cmdpos < (maxdelta - indelta - 1))
    {
      indelta += cmdpos;
      cptr = command;
      while (cmdpos--)
        *delta++ = *cptr++;
    }
    else
    {
      delta[indelta++] = 'D';  /* this delta buffer is done */
      bailout = 1;  /* not enough space left, abort & come back later */
      saved_position = position;
      saved_cmdpos = cmdpos;
    }
  }

/* terminate the buffer */
```

```
  *delta = 'D';
  indelta++;

  return (indelta);
}

int SpImageAddData(display,image,data)
Display display;
Image image;
unsigned char *data;
{
  int deltalength,i;

  if (!image->displayValid)
  {
    write(display,"@",1);  /* tell the SPD that this is a binary data block */
    write(display,data,image->buflength); /* send the data over */
    write(display,"\n",1);  /* terminate the data block  */
    image->displayValid = TRUE;
#ifdef SPDEBUG
fprintf(stderr,"wrote [%d] bytes\n",image->buflength);
#endif
  }
  else
  {
    deltalength = SpImageGenDelta(image->data,data,image->buflength,-
deltabuffer,DELTABUFSIZE);
    write(display,"$",1);  /* tell the SPD that this is a binary data block */
    write(display,&deltalength,8);  /* tell the SPD that this block has deltalength bytes */
    write(display,deltabuffer,deltalength); /* send the data over */
    write(display,"\n",1);  /* tell the SPD that this is a binary data block */

#ifdef SPDEBUG
fprintf(stderr,"Compression enabled: [%d]\n",deltalength);
SpDumpDelta(deltabuffer);
#endif
  }

  /* copy the new to the old */
  for (i = 0; i < image->buflength; i++)
    image->data[i] = *data++;

  return (0);
}
```

```
/
***************************************************************************
*****************
*  Lgraph routines                                           *
*                                                          *
*                                                          *
***************************************************************************
*****************/

Graph SpLgraphOpen(display,minX,minY,maxX,maxY,xlabel,ylabel,title)
Display display;
double minX,minY,maxX,maxY;
char *xlabel,*ylabel,*title;
{
  char buffer[128];
  char *cptr;
  Graph handle;

  if (!xlabel)
    xlabel = "X";

  if (!ylabel)
    ylabel = "Y";

  if (!title)
    title = "SP";

  /* open a graph */
  sprintf(buffer,"#lgraphm %g %g %g %g\"%s\" \"%s\" \"%s\"\n",minX,minY,maxX,max-
Y,xlabel,ylabel,title);
  write(display,buffer,strlen(buffer));

  /* wait for confirmation about the graph handle.
     This requires a round-trip to the server, so
     it could take a while.
  */
  cptr = buffer;
  read(display,cptr,1);
  while (*cptr != '\n')
  {
    cptr++;
    read(display,cptr,1);
  }
  *cptr = '\0';
  handle = atoi(buffer);
  return (handle);
}

SpLgraphClose(display,handle)
Display display;
Graph handle;
{
  char tmpbuf[64];
  /* close the graph */
  sprintf(tmpbuf,"#quit %d\n",handle);  /* do nothing for now */
  write(display,tmpbuf,strlen(tmpbuf));  /* do nothing for now */
}

SpLgraphGridlines(display,handle,gridx,gridy)
Display display;
Graph handle;
int gridx;
int gridy;
{
  char buffer[64];

  /* Turn X gridlines on/off */
  sprintf(buffer,"#xgridlines %s %d\n",(gridx) ? "TRUE" : "FALSE",handle);
  write(display,buffer,strlen(buffer));

  /* Turn Y gridlines on/off */
  sprintf(buffer,"#ygridlines %s %d\n",(gridy) ? "TRUE" : "FALSE",handle);
  write(display,buffer,strlen(buffer));
}

SpLgraphDivisions(display,handle,divx,divy)
Display display;
Graph handle;
int divx;
int divy;
{
  char buffer[64];

  /* set the number of divisions on the X axis */
  if (divx >= 0)  /* only set if not "ignore" */
  {
    sprintf(buffer,"#xdivisions %d %d\n",divx,handle);
    write(display,buffer,strlen(buffer));
  }
  /* set the number of divisions on the Y axis */
  if (divy >= 0)  /* only set if not "ignore" */
  {
    sprintf(buffer,"#ydivisions %d %d\n",divy,handle);
    write(display,buffer,strlen(buffer));
  }
}

SpShutdown(display)
Display display;
{
  write(display,"#shutdown\n",10);  /* tell the server to close the connection & quit */
  close(display);
}

SpLgraphAddData(display,handle,x,yv,yvc)
Display display;
Graph handle;
float x;
float yv[];
int yvc;
{
  char buffer[128];
  int i;

  /* build up the buffer to be sent out */
  sprintf(buffer,"#%d %g",handle,x);
  for (i = 0; i < yvc; i++)
    sprintf(&buffer[strlen(buffer)]," %g",yv[i]);
  strcat(buffer,"\n");

  /* send it out over the network */
  /* fprintf(stdout,"%s",buffer); */
  write(display,buffer,strlen(buffer));
}

SpLgraphAddDataD(display,handle,x,yv,yvc)
Display display;
Graph handle;
double x;
double yv[];
int yvc;
{
  char buffer[128];
  int i;

  /* build up the buffer to be sent out */
  sprintf(buffer,"#%d %g",handle,x);
  for (i = 0; i < yvc; i++)
    sprintf(&buffer[strlen(buffer)]," %g",yv[i]);
  strcat(buffer,"\n");

  /* send it out over the network */
  /* fprintf(stdout,"%s",buffer); */
  write(display,buffer,strlen(buffer));
}

#ifdef CRAY

/* Binary transmission supported from Cray to Sun
*/

SpLgraphAddBinaryData(display,handle,x,yv,yvc)
Display display;
Graph handle;
float x;
float yv[];
int yvc;
{
  char *cptr;
  int i;
  struct
  {
    char header;
    char graph;
    char yvc;
    char filler[5]; /* nothing stored here */
    unsigned int x;
    unsigned int yv[32];
  } buffer;

  buffer.header = '!';
  buffer.graph = (char)handle;
  buffer.yvc = (char)yvc;
  buffer.x = crayFloatToIEEE(x);

#ifdef SPDEBUG
printf("%016lx ",buffer.x);
  for (i = 0; i < yvc; i++)
  {
    buffer.yv[i] = crayFloatToIEEE(yv[i]);
printf("%016lx ",buffer.yv[i]);
  }
printf("\n");
#else
  for (i = 0; i < yvc; i++)
    buffer.yv[i] = crayFloatToIEEE(yv[i]);
#endif

  /* send it out over the network */
  write(display,&buffer,16 + 8 * i);
  write(display,"\n",1); /* temporary record terminator */
}
#endif

#ifdef SPDEBUG
mwrite(handle,buffer,length)
int handle;
char *buffer;
int length;
{
  int i;

  for (i = 0; i < length; i++)
    printf("%02x ",buffer[i]);
  printf("\n");

  write(handle,buffer,length);
}
#endif

/*
  crayFloatToIEEE()

  Takes as input a 64-bit Cray format floating point number, and
  returns a properly formatted IEEE 64-bit binary value stored in
  a 64-bit unsigned integer.
*/
#ifdef CRAY
unsigned int crayFloatToIEEE(f)
float f;
{
  unsigned int i, *iptr;
  unsigned int sign;
  unsigned int exponent;
  unsigned int coeff;
  unsigned int new;

  /* special case at zero */
  if (f == 0.0)
    return (0x0000000000000000);

  /* break the old value down into it's components */
  iptr = (unsigned int *)&f;
  i = *iptr;
  sign = i & 0x8000000000000000;
  exponent = ((i & 0x7fff000000000000) >> 48) - 0x4000;

  /* modify the coefficient so it fits the 1.f format */
  coeff = ((i & 0x0000ffffffffffff) << 1) & 0x0000ffffffffffff;
  exponent--;

  /* check for exponent overflow */
  exponent += 1023;

  if (exponent > 0x7fe || exponent < 1)
    return (0x7ff0000000000000);  /* not a number */

  /* put the new value together from scratch */
  new = sign | (coeff << 4);
  new |= (exponent) << 52;

  return (new);
}
#endif
```

APPENDIX I

VISUALIZATION WIDGET SOURCES

```
/*
  Lgraph.H

  Change History
      3-Nov-90/mwl - Original Issue
*/

#ifndef _XLgraph_h
#define _XLgraph_h

/***********************************************************************
 *
 * Lgraph Widget
 *
 ***********************************************************************/

#include <X11/Core.h>

/* Resources:

 Name              Class          RepType        Default Value
 ----              -----          -------        -------------
 background        BackgroundPixel  XtDefaultBackground
 bitmap            Pixmap         Pixmap         None
 border            BorderColorPixel              XtDefaultForeground
 borderWidth       BorderWidthDimension1
 callback          Callback              Pointer        NULL
 cursor            Cursor         Cursor         None
 destroyCallback   CallbackPointerNULL
 font              Font           XFontStruct*XtDefaultFont
 foreground        ForegroundPixel               XtDefaultForeground
 height            Height         Dimensiontext height
 insensitiveBorder  InsensitivePixmapGray
 internalHeight    HeightDimension2
 internalWidth     WidthDimension4
 justify           Justify        XtJustifyXtJustifyCenter
 label             Label          String         NULL
 mappedWhenManaged  MappedWhenManagedBooleanTrue
 resize            Resize         Boolean        True
 sensitive         Sensitive      Boolean        True
 width             Width          Dimensiontext width
 x                 Position       Position0
 y                 Position       Position0
 xdividers         Xdividers         Boolean        FALSE
 ydividers         Ydividers         Boolean        FALSE
```

```
xdivisions     Xdivisions     int       5
ydivisions     Ydivisions     int       5
maxX           MaxX           int       0
maxY           MaxY           int       0
minX           MinX           int       0
minY           MinY           int       0
xlogscale      Xlogscale      Boolean      FALSE
ylogscale      Ylogscale      Boolean      FALSE
color0-7       Color0-7       Pixel     XtDefaultForeground
shiftMode       ShiftMode      int        10
xminor         Xminor         int       5
yminor         Yminor         int       5
*/

/* Private Atoms */
#define XtNxdividers "xdividers"
#define XtCXdividers "Xdividers"
#define XtNydividers "ydividers"
#define XtCYdividers "Ydividers"
#define XtNxdivisions "xdivisions"
#define XtCXdivisions "Xdivisions"
#define XtNydivisions "ydivisions"
#define XtCYdivisions "Ydivisions"
#define XtNmaxX "maxX"
#define XtCMaxX "MaxX"
#define XtNmaxY "maxY"
#define XtCMaxY "MaxY"
#define XtNminX "minX"
#define XtCMinX "MinX"
#define XtNminY "minY"
#define XtCMinY "MinY"
#define XtNxlogscale "xlogscale"
#define XtCXlogscale "Xlogscale"
#define XtNylogscale "ylogscale"
#define XtCYlogscale "Ylogscale"
#define XtNcolor0 "color0"
#define XtCColor0 "Color0"
#define XtNcolor1 "color1"
#define XtCColor1 "Color1"
#define XtNcolor2 "color2"
#define XtCColor2 "Color2"
#define XtNcolor3 "color3"
#define XtCColor3 "Color3"
#define XtNcolor4 "color4"
#define XtCColor4 "Color4"
```

```
#define XtNcolor5 "color5"
#define XtCColor5 "Color5"
#define XtNcolor6 "color6"
#define XtCColor6 "Color6"
#define XtNcolor7 "color7"
#define XtCColor7 "Color7"
#define XtNshiftMode "shiftMode"
#define XtCShiftMode "ShiftMode"
#define XtNxlabel "xlabel"
#define XtCXlabel "Xlabel"
#define XtNylabel "ylabel"
#define XtCYlabel "Ylabel"
#define XtNtitle "title"
#define XtCTitle "Title"
#define XtNlabelFont "labelFont"
#define XtCLabelFont "LabelFont"
#define XtNxminor "xminor"
#define XtCXminor "Xminor"
#define XtNyminor "yminor"
#define XtCYminor "Yminor"
#define XtNticksize "ticksize"
#define XtCTicksize "Ticksize"
#define XtNmticksize "mticksize"
#define XtCMticksize "Mticksize"

extern WidgetClass     lgraphWidgetClass;

typedef struct _LgraphClassRec   *LgraphWidgetClass;
typedef struct _LgraphRec        *LgraphWidget;


#endif /* _XLgraph_h */
/* DON'T ADD STUFF AFTER THIS */
```

```
/*
  LgraphP.h - Private definitions for Lgraph widget

  Change History
      3-Nov-90/mwl - Original Issue
 */

#ifndef _XLgraphP_h
#define _XLgraphP_h

#include "Lgraph.h"
#include <X11/CoreP.h>

/*********************************************************************
 *
 * Lgraph Widget Private Data
 *
 *********************************************************************/

typedef struct LDATA_TAG
{
struct LDATA_TAG *nextdata;
double x;
double *yv;
int yvc;
} LDATA;


/**********************************
 *
 *  Class structure
 *
 **********************************/


  /* New fields for the Lgraph widget class record */
typedef struct _LgraphClass
 {
  int makes_compiler_happy;  /* not used */
 } LgraphClassPart;

  /* Full class record declaration */
typedef struct _LgraphClassRec {
   CoreClassPartcore_class;
   LgraphClassPart    lgraph_class;
```

```
} LgraphClassRec;

extern LgraphClassRec lgraphClassRec;

/*************************************
 *
 * Instance (widget) structure
 *
 *************************************/

  /* New fields for the Lgraph widget record */

typedef struct {
  /* resources */
  XtCallbackList callbacks;
  float           minX; /* lowest X value */
  float           minY; /* lowest Y value */
  float           maxX; /* highest X value */
  float           maxY; /* highest Y value */
  Boolean         xdividers;
  Boolean         ydividers;
  int             xdivisions;
  int             ydivisions;
  int             xminor;
  int             yminor;
  int             ticksize;
  int             mticksize;
  Boolean         xlogscale;
  Boolean         ylogscale;
  Pixel           foreground;
  Pixel           color0;
  Pixel           color1;
  Pixel           color2;
  Pixel           color3;
  Pixel           color4;
  Pixel           color5;
  Pixel           color6;
  Pixel           color7;
  int             shiftMode;
  char            *title;
  char            *titlestring;
  char            *xlabel;
  char            *ylabel;
  XFontStruct     *labelFont;
  XFontStruct     *font;
  /* private state */
  LDATA  *ldata; /* data area */
  LDATA  *lastdata; /* pointer to last data point in the list */
  Pixmap  gray_pixmap;
  GC      normal_GC;
  GC      inverse_GC;
  GC      grid_GC;
  GC      color_GC[8];
  Boolean rtOK;
  float   minX;
  float   minY;
  float   maxX;
  float   maxY;
  int     margin;
  int     height;
  float   xscale;
  float   yscale;
} LgraphPart;


/* Full widget declaration */
typedef struct _LgraphRec {
  CorePart     core;
  LgraphPart   lgraph;
} LgraphRec;

#endif /* _XLgraphP_h */
```

```
/*
 Lgraph.c - Lgraph widget

 Change History
      3-Nov-90/mwl - Original Issue
 */

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <math.h>
#include <X11/Xlib.h>
#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include "LgraphP.h"

/****************************************************************
 *
 * Full class record constant
 *
 ****************************************************************/

/* Private Data */

static char defaultTranslations[] =
   "<Btn1Down>:notify()\n\
    <Btn3Down>:set()";

#define offset(field) XtOffset(LgraphWidget, field)

static XtResource resources[] = {
  {XtNcallback, XtCCallback, XtRCallback, sizeof(XtPointer),
    offset(lgraph.callbacks), XtRCallback, (XtPointer)NULL},
  {XtNxdividers, XtCBoolean, XtRBoolean, sizeof(Boolean),
    offset(lgraph.xdividers), XtRImmediate, (caddr_t)FALSE},
  {XtNydividers, XtCBoolean, XtRBoolean, sizeof(Boolean),
    offset(lgraph.ydividers), XtRImmediate, (caddr_t)FALSE},
  {XtNxdivisions, XtCXdivisions, XtRInt, sizeof(int),
    offset(lgraph.xdivisions), XtRImmediate, (caddr_t)-1},
  {XtNydivisions, XtCYdivisions, XtRInt, sizeof(int),
    offset(lgraph.ydivisions), XtRImmediate, (caddr_t)-1},
  {XtNxlogscale, XtCBoolean, XtRBoolean, sizeof(Boolean),
    offset(lgraph.xlogscale), XtRImmediate, (caddr_t)FALSE},
  {XtNylogscale, XtCBoolean, XtRBoolean, sizeof(Boolean),
    offset(lgraph.ylogscale), XtRImmediate, (caddr_t)FALSE},
  {XtNxlabel,XtCXlabel, XtRString, sizeof(String),
    offset(lgraph.xlabel), XtRString, "X" },
  {XtNylabel,XtCYlabel, XtRString, sizeof(String),
    offset(lgraph.ylabel), XtRString, "Y" },
  {XtNtitle,XtCTitle, XtRString, sizeof(String),
    offset(lgraph.title), XtRString, "" },
  {XtNstring,XtCString, XtRString, sizeof(String),
    offset(lgraph.titlestring), XtRString, "" },
  {XtNcolor0,XtCColor0, XtRPixel, sizeof(Pixel),
    offset(lgraph.color0), XtRString, "XtDefaultForeground" },
  {XtNforeground,XtCForeground, XtRPixel, sizeof(Pixel),
    offset(lgraph.foreground), XtRString, "black" },
  {XtNcolor1,XtCColor1, XtRPixel, sizeof(Pixel),
    offset(lgraph.color1), XtRString, "red" },
  {XtNcolor2,XtCColor2, XtRPixel, sizeof(Pixel),
    offset(lgraph.color2), XtRString, "green" },
  {XtNcolor3,XtCColor3, XtRPixel, sizeof(Pixel),
    offset(lgraph.color3), XtRString, "blue" },
  {XtNcolor4,XtCColor4, XtRPixel, sizeof(Pixel),
    offset(lgraph.color4), XtRString, "orange" },
  {XtNcolor5,XtCColor5, XtRPixel, sizeof(Pixel),
    offset(lgraph.color5), XtRString, "magenta" },
  {XtNcolor6,XtCColor6, XtRPixel, sizeof(Pixel),
    offset(lgraph.color6), XtRString, "brown" },
  {XtNcolor7,XtCColor7, XtRPixel, sizeof(Pixel),
    offset(lgraph.color7), XtRString, "navy" },
  {XtNlabelFont,XtCLabelFont,XtRFontStruct,sizeof(XFontStruct *),
    offset(lgraph.labelFont),XtRString, "vtsingle" },
  {XtNfont,XtCLabelFont,XtRFontStruct,sizeof(XFontStruct *),
    offset(lgraph.font),XtRString, "vtsingle" },
  {XtNxminor,XtCXminor,XtRInt,sizeof(int),
    offset(lgraph.xminor),XtRImmediate, (caddr_t)5 },
  {XtNyminor,XtCYminor,XtRInt,sizeof(int),
    offset(lgraph.yminor),XtRImmediate, (caddr_t)5 },
  {XtNticksize,XtCTicksize,XtRInt,sizeof(int),
    offset(lgraph.ticksize),XtRImmediate, (caddr_t)10 },
  {XtNmticksize,XtCMticksize,XtRInt,sizeof(int),
    offset(lgraph.mticksize),XtRImmediate, (caddr_t)6 },
  {XtNshiftMode,XtCShiftMode,XtRInt,sizeof(int),
    offset(lgraph.shiftMode),XtRImmediate, (caddr_t)10 }
};
#undef offset

static Boolean SetValues();
static void Initialize(), Redisplay(), Set(), Reset(), Notify(), Unset();
static void Destroy(), PaintLgraphWidget();
static void ClassInitialize();

static XtActionsRec actionsList[] =
{
 {"set",Set},
 {"notify",Notify},
 {"reset",Reset},
 {"unset",Unset},
};

#define SuperClass ((CoreWidgetClass)&coreClassRec)

LgraphClassRec lgraphClassRec = {
 {
  (WidgetClass) SuperClass,/* superclass */
  "Lgraph",/* class_name */
  sizeof(LgraphRec),/* size */
  NULL,/* class_initialize */
  NULL,/* class_part_initialize */
  FALSE,/* class_inited */
  Initialize,/* initialize */
  NULL,/* initialize_hook */
  XtInheritRealize,/* realize */
  actionsList,/* actions */
  XtNumber(actionsList),/* num_actions */
  resources,/* resources */
  XtNumber(resources),/* resource_count */
  NULLQUARK,/* xrm_class */
  FALSE,/* compress_motion */
  TRUE,/* compress_exposure */
  TRUE,/* compress_enterleave   */
  FALSE,/* visible_interest */
  Destroy,/* destroy */
  XtInheritResize,/* resize */
  Redisplay,/* expose */
  SetValues,/* set_values */
  NULL,/* set_values_hook */
  XtInheritSetValuesAlmost,/* set_values_almost */
  NULL,/* get_values_hook */
  NULL,/* accept_focus */
  XtVersion,/* version */
  NULL,/* callback_private */
  defaultTranslations,/* tm_table */
  XtInheritQueryGeometry,/* query_geometry */
  XtInheritDisplayAccelerator,/* display_accelerator */
  NULL/* extension */
 }, /* CoreClass fields initialization */
 {
  0,                        /* field not used   */
 }, /* LgraphClass fields initialization */
};

 /* for public consumption */
WidgetClass lgraphWidgetClass = (WidgetClass) &lgraphClassRec;

/****************************************************************
 *
 * Private Procedures
 *
 ****************************************************************/

static GC
Get_GC(cbw, fg, bg, font, linetype, dashtype)
LgraphWidget cbw;
Pixel fg, bg;
Font font;
int linetype;
char dashtype;
{
 XGCValuesvalues;

 values.foreground  = fg;
 values.background= bg;
 values.font= font;
 values.cap_style = CapProjecting;
 values.line_width  = 0;

 if (linetype != LineSolid)
 {
   values.line_style = linetype;
   values.dash_offset = 0;
   values.dashes = dashtype;

   return XtGetGC((Widget)cbw,
(GCForeground|GCBackground|GCFont|GCLineWidth|GCCapStyle|GCLineStyle|
     GCDashOffset|GCDashList),
&values);
 }
 else
```

```
   return XtGetGC((Widget)cbw,
 (GCForeground|GCBackground|GCFont|GCLineWidth|GCCapStyle),
 &values);
}


/* ARGSUSED */
static void
Initialize(request, new, args, num_args)
Widget request, new;
ArgList args;/* unused */
Cardinal *num_args;/* unused */
{
 LgraphWidget cbw = (LgraphWidget) new;

 /* set a reasonable desired size */
 cbw->core.width = 500;
 cbw->core.height = 350;

 /* get GCs so we can draw */
 cbw->lgraph.normal_GC = Get_GC(cbw, cbw->lgraph.foreground, cbw->core.back-
ground_pixel,
    cbw->lgraph.font->fid,LineSolid,0);
 cbw->lgraph.inverse_GC = Get_GC(cbw, cbw->core.background_pixel, cbw->lgraph.-
foreground,
    cbw->lgraph.font->fid,LineSolid,0);
 cbw->lgraph.grid_GC = Get_GC(cbw, cbw->lgraph.foreground, cbw->core.background_-
pixel,
    cbw->lgraph.labelFont->fid,LineDoubleDash,1);
 cbw->lgraph.color_GC[0] = Get_GC(cbw, cbw->lgraph.color0,cbw->core.background_-
pixel,
    cbw->lgraph.font->fid,LineSolid,0);
 cbw->lgraph.color_GC[1] = Get_GC(cbw, cbw->lgraph.color1,cbw->core.background_-
pixel,
    cbw->lgraph.font->fid,LineSolid,0);
 cbw->lgraph.color_GC[2] = Get_GC(cbw, cbw->lgraph.color2,cbw->core.background_-
pixel,
    cbw->lgraph.font->fid,LineSolid,0);
 cbw->lgraph.color_GC[3] = Get_GC(cbw, cbw->lgraph.color3,cbw->core.background_-
pixel,
    cbw->lgraph.font->fid,LineSolid,0);
 cbw->lgraph.color_GC[4] = Get_GC(cbw, cbw->lgraph.color4,cbw->core.background_-
pixel,
    cbw->lgraph.font->fid,LineSolid,0);
 cbw->lgraph.color_GC[5] = Get_GC(cbw, cbw->lgraph.color5,cbw->core.background_-
pixel,
    cbw->lgraph.font->fid,LineSolid,0);
  cbw->lgraph.color_GC[6] = Get_GC(cbw, cbw->lgraph.color6,cbw->core.background_-
pixel,
    cbw->lgraph.font->fid,LineSolid,0);
  cbw->lgraph.color_GC[7] = Get_GC(cbw, cbw->lgraph.color7,cbw->core.background_-
pixel,
    cbw->lgraph.font->fid,LineSolid,0);

  cbw->lgraph.ldata = NULL;  /* starts with no graph data */

  cbw->lgraph.iminX = 0.0;
  cbw->lgraph.imaxX = 0.0;
  cbw->lgraph.iminY = 0.0;
  cbw->lgraph.imaxY = 0.0;
  cbw->lgraph.minX = 0.0;
  cbw->lgraph.maxX = 0.0;
  cbw->lgraph.minY = 0.0;
  cbw->lgraph.maxY = 0.0;

  /* do not start real-time representations until the graph has been properly realized */
  cbw->lgraph.rtOK = FALSE;
}


/***************************
*
* Action Procedures
*
***************************/

/* ARGSUSED */
static void
Set(w,event,params,num_params)
Widget w;
XEvent *event;
String *params;/* unused */
Cardinal *num_params;/* unused */
{
  LgraphWidget cbw = (LgraphWidget)w;

  XtCallCallbacks(w,XtNcallback,(caddr_t)event);
}

/* ARGSUSED */
```

```
static void
Unset(w,event,params,num_params)
Widget w;
XEvent *event;
String *params;/* unused */
Cardinal *num_params;
{
 LgraphWidget cbw = (LgraphWidget)w;

}

/* ARGSUSED */
static void
Reset(w,event,params,num_params)
Widget w;
XEvent *event;
String *params;/* unused */
Cardinal *num_params;  /* unused */
{
 LgraphWidget cbw = (LgraphWidget)w;
}

/* ARGSUSED */
static void
Notify(w,event,params,num_params)
Widget w;
XEvent *event;
String *params;/* unused */
Cardinal *num_params;/* unused */
{
 LgraphWidget cbw = (LgraphWidget)w;
 LDATA *data;
 int i;

 printf("minX: %g minY: %g maxX: %g maxY: %g\n",cbw->lgraph.minX,
                              cbw->lgraph.minY,
                              cbw->lgraph.maxX,
                              cbw->lgraph.maxY);

 printf("iminX: %g iminY: %g imaxX: %g imaxY: %g\n",cbw->lgraph.iminX,
                              cbw->lgraph.iminY,
                              cbw->lgraph.imaxX,
                              cbw->lgraph.imaxY);

 for (data = cbw->lgraph.ldata; data; data = data->nextdata)
```

```
  {
    printf("%g",data->x);

    for (i = 0; i < data->yvc; i++)
    {
      printf(" %f",data->yv[i]);
    }

    printf("\n");
  }
}

/*
 * Repaint the widget window
 */

/* ARGSUSED */
static void
Redisplay(w, event, region)
Widget w;
XExposeEvent *event;
Region region;
{
  if (event)
  {
    /* only clear the display if the whole window needs to be redrawn */
    if (event->x == 0 && event->y == 0)
      XClearWindow(XtDisplay(w),XtWindow(w));
  }

  lgraphDisplay(w);
}


static void
Destroy(w)
Widget w;
{
  int i;

  LgraphWidget cbw = (LgraphWidget) w;

  /* free up data area memory */
  lgraphFree(cbw);
 /* so Label can release it */
 XtReleaseGC( w, cbw->lgraph.normal_GC);
 XtReleaseGC( w, cbw->lgraph.inverse_GC);

 for (i = 0; i < 8; i++)
   XtReleaseGC( w, cbw->lgraph.color_GC[i]);
}

/*
 * Set specified arguments into widget
 */

/* ARGSUSED */
static Boolean
SetValues (current, request, new)
Widget current, request, new;
{
 LgraphWidget oldcbw = (LgraphWidget) current;
 LgraphWidget cbw = (LgraphWidget) new;
 Boolean redisplay = False;

 if ( (oldcbw->lgraph.foreground!= cbw->lgraph.foreground) ||
     (oldcbw->core.background_pixel != cbw->core.background_pixel) ||
     (oldcbw->lgraph.font != cbw->lgraph.font) )
 {
   XtReleaseGC(new, cbw->lgraph.inverse_GC);
   XtReleaseGC(new, cbw->lgraph.normal_GC);

   cbw->lgraph.normal_GC = Get_GC(cbw, cbw->lgraph.foreground,
   cbw->core.background_pixel);
   cbw->lgraph.inverse_GC = Get_GC(cbw, cbw->core.background_pixel,
    cbw->lgraph.foreground);

   redisplay = True;
 }


 return (redisplay);
}

/*
 * Private subroutines for manipulating the database
 *
 */
```

```
#define GRAPHBORDER 40

int lgraphAddData(w,x,yv,yvc)
LgraphWidget w;
double x;  /* X value */
double *yv; /* Y vector */
int yvc;   /* length of Y vector */
{
  LDATA *data,*prevdata;
  int i;
  int x1,y1,x2,y2;

  prevdata = data;
  data = (LDATA *)malloc(sizeof(LDATA));
  data->x = x;
  data->yv = (double *)malloc(sizeof(double) * yvc);
  for (i = 0; i < yvc; i++)
    data->yv[i] = yv[i];
  data->yvc = yvc;

  if (!w->lgraph.ldata)
    w->lgraph.ldata = data;
  else
    w->lgraph.lastdata->nextdata = data;  /* apply link within list */

  data->nextdata = NULL;   /* finish the list with a NULL terminator */

  if (x > w->lgraph.maxX)
    w->lgraph.maxX = x;
  if (x < w->lgraph.minX)
    w->lgraph.minX = x;

  for (i = 0; i < yvc; i++)
  {
    if (yv[i] > w->lgraph.maxY)
      w->lgraph.maxY = yv[i];
    if (yv[i] < w->lgraph.minY)
      w->lgraph.minY = yv[i];
  }

  /* if the widget is already realized, and this is additional info to display
    draw more.
  */
  if (w->lgraph.rtOK && w->lgraph.lastdata && data)
  {
    /* if graph tolerances have been breached, just redisplay */
    if (w->lgraph.maxX > w->lgraph.imaxX || w->lgraph.maxY > w->lgraph.imaxY || w-
>lgraph.minY < w->lgraph.iminY ||
        w->lgraph.minX < w->lgraph.iminX)
    {
      XClearWindow(XtDisplay(w),XtWindow(w));
      rescale(w);
      lgraphDisplay(w);
    }
    else  /* this line will still fall within our graph area, so draw it */
    {
      for (i = 0; i < data->yvc; i++)
      {
      x1 = w->lgraph.lmargin + w->lgraph.xscale * (w->lgraph.lastdata->x - w->lgraph.im-
inX) + GRAPHBORDER / 2;
        x2 = w->lgraph.lmargin + w->lgraph.xscale * (data->x - w->lgraph.iminX) +
GRAPHBORDER / 2;
        y1 = w->lgraph.height - w->lgraph.yscale * (w->lgraph.lastdata->yv[i] - w->lgra-
ph.iminY) - GRAPHBORDER / 2;
        y2 = w->lgraph.height - w->lgraph.yscale * (data->yv[i] - w->lgraph.iminY) -
GRAPHBORDER / 2;
        XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.color_GC[i%8],x1,y1,x2,y2);
      }
    }
  }

  /* finalize the links to the data areas */
  w->lgraph.lastdata = data;    /* add end-link to get back to it fast */
}

static rescale(w)
LgraphWidget w;
{
  if (w->lgraph.shiftMode)
  {
    if (w->lgraph.maxX > w->lgraph.imaxX)
      w->lgraph.imaxX = w->lgraph.maxX * (1.0 + w->lgraph.shiftMode / 100.0);
    if (w->lgraph.maxY > w->lgraph.imaxY)
      w->lgraph.imaxY = w->lgraph.maxY * (1.0 + w->lgraph.shiftMode / 100.0);
    if (w->lgraph.minY < w->lgraph.iminY)
      w->lgraph.iminY = w->lgraph.minY * (1.0 + w->lgraph.shiftMode / 100.0);
    if (w->lgraph.minX < w->lgraph.iminX)
      w->lgraph.iminX = w->lgraph.minX * (1.0 + w->lgraph.shiftMode / 100.0);
  }
}
```

```
lgraphSetLimits(w,minx,miny,maxx,maxy,mode)
LgraphWidget w;
float minx,miny,maxx,maxy;
int mode;
{
  w->lgraph.iminX = minx;
  w->lgraph.iminY = miny;
  w->lgraph.imaxX = maxx;
  w->lgraph.imaxY = maxy;
  w->lgraph.shiftMode = mode;
}

#define MTICKSIZE w->lgraph.mticksize
#define TICKSIZE w->lgraph.ticksize
#define LABELSEPARATION 5

static int lgraphDisplay(w)
LgraphWidget w;
{
  int height,width;
  int depth;
  int x;
  int y;
  int bwidth;
  Window win;
  int x1,x2,y1,y2;
  LDATA *data;
  int i;
  float value;
  float minX,maxX,minY,maxY;
  double magnitude;
  char buffer[64];
  int xdivisions,ydivisions;
  int labelwidth;
  float xscale,yscale;
  int lmargin;
  int maxyw;

  w->lgraph.rtOK = TRUE;  /* valid data will be entered for graph scaling (real-time pro-
cessing is OK) */

  /* determine a reasonable number of divisions if necessary */
  if (w->lgraph.xdivisions == -1)
    xdivisions = 5;
  else
    xdivisions = w->lgraph.xdivisions;

  if (w->lgraph.ydivisions == -1)
    ydivisions = 4;
  else
    ydivisions = w->lgraph.ydivisions;

  /* make sure that the minimums and maximums are reasonable */
  minX = w->lgraph.iminX;
  maxX = w->lgraph.imaxX;

  minY = w->lgraph.iminY;
  maxY = w->lgraph.imaxY;

  if (minY == maxY)
  {
    minY = w->lgraph.iminY = minY - .01;
    maxY = w->lgraph.imaxY = maxY - .01;
  }

  /* find the longest of the y value labels */
  for (maxyw = 0, i = 0; i <= ydivisions; i++)
  {
    sprintf(buffer,"%.4G", (maxY - minY) * i / ydivisions + minY);
    if (strlen(buffer) > maxyw)
      maxyw = strlen(buffer);
  }
  if (maxyw > 25)  /* don't overshoot the '8' buffer */
    maxyw = 25;

  /* calculate the necessary margins using font information */
  labelwidth = XTextWidth(w->lgraph.font,w->lgraph.ylabel,strlen(w->lgraph.ylabel)) +
LABELSEPARATION;
  lmargin = w->lgraph.lmargin = labelwidth + XTextWidth(w->lgraph.-
font,".8888888888888888888888888",maxyw) +
        LABELSEPARATION;

  /* make sure the widths and heights are correct */
  height = w->lgraph.height = w->core.height - w->lgraph.font->ascent - w->lgraph.font-
>descent - 5;
  width = w->core.width - lmargin - 15;

  /* scaling information */
  yscale = w->lgraph.yscale = fabs((height - GRAPHBORDER) / (maxY - minY));
  xscale = w->lgraph.xscale = fabs((width - GRAPHBORDER) / (maxX - minX));

  /* Draw Graph Surround Box */
  XDrawRectangle(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
           lmargin + GRAPHBORDER / 2,
           GRAPHBORDER / 2,
           width - GRAPHBORDER,
           height - GRAPHBORDER);

  /* Draw Title */
  XDrawString(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
        w->core.width / 2 - XTextWidth(w->lgraph.font,w->lgraph.titlestring,strlen(w-
>lgraph.title)) / 2,
        w->lgraph.font->ascent,
        w->lgraph.title,
        strlen(w->lgraph.titlestring));

  /* Draw Minor X Tick Marks */
  for (i = 1; w->lgraph.xminor && i < xdivisions * w->lgraph.xminor; i++)
  {
    /* draw axis tick marks */
    XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.grid_GC,
         (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions /
w->lgraph.xminor),
         (int)(fabs(maxY) * yscale + GRAPHBORDER / 2 + MTICKSIZE / 2),
         (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions /
w->lgraph.xminor),
         (int)(fabs(maxY) * yscale + GRAPHBORDER / 2 - MTICKSIZE / 2));

    /* draw base tick marks */
    XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.grid_GC,
         (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions /
w->lgraph.xminor),
         (int)(height - GRAPHBORDER / 2),
         (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions /
w->lgraph.xminor),
         (int)(height - GRAPHBORDER / 2 - MTICKSIZE));

    /* draw top tick marks */
    XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.grid_GC,
         (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions /
w->lgraph.xminor),
         (int)(GRAPHBORDER / 2),
         (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions /
w->lgraph.xminor),
        (int)(GRAPHBORDER / 2 + MTICKSIZE));
  }

  /* Draw Minor Y Tick Marks */
  for (i = 1; w->lgraph.yminor && i < ydivisions * w->lgraph.yminor; i++)
  {
    /* Draw Y axis tick marks */
    XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.grid_GC,
        (int)(lmargin + GRAPHBORDER / 2 + fabs(minX) * xscale + MTICKSIZE / 2),
      (int)(height - i * (maxY - minY) * yscale / ydivisions / w->lgraph.yminor - GRAPH-
BORDER / 2),
        (int)(lmargin + GRAPHBORDER / 2 + fabs(minX) * xscale - MTICKSIZE / 2),
      (int)(height - i * (maxY - minY) * yscale / ydivisions / w->lgraph.yminor - GRAPH-
BORDER / 2));

    /* Draw left tick marks */
    XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.grid_GC,
        (int)(lmargin + GRAPHBORDER / 2),
      (int)(height - i * (maxY - minY) * yscale / ydivisions / w->lgraph.yminor - GRAPH-
BORDER / 2),
        (int)(lmargin + GRAPHBORDER / 2 + MTICKSIZE),
      (int)(height - i * (maxY - minY) * yscale / ydivisions / w->lgraph.yminor - GRAPH-
BORDER / 2));

    /* Draw right tick marks */
    XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.grid_GC,
        (int)(lmargin + width - GRAPHBORDER / 2),
      (int)(height - i * (maxY - minY) * yscale / ydivisions / w->lgraph.yminor - GRAPH-
BORDER / 2),
        (int)(lmargin + width - GRAPHBORDER / 2 - MTICKSIZE),
      (int)(height - i * (maxY - minY) * yscale / ydivisions / w->lgraph.yminor - GRAPH-
BORDER / 2));
  }

  /* Draw Major X Tick Marks */
  for (i = 0; i <= xdivisions; i++)
  {
    if (i != 0 && i != xdivisions)
    {
      if (w->lgraph.xdividers)
      {
        XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.grid_GC,
          (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions),
            (int)(height - GRAPHBORDER / 2),
          (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions),
            (int)(GRAPHBORDER / 2));
      }

      /* draw axis tick marks */
      XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
           (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions),
            (int)(fabs(maxY) * yscale + GRAPHBORDER / 2 + TICKSIZE / 2),
           (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions),
            (int)(fabs(maxY) * yscale + GRAPHBORDER / 2 - TICKSIZE / 2));

      /* draw base tick marks */
      XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
           (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions),
            (int)(height - GRAPHBORDER / 2),
           (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions),
            (int)(height - GRAPHBORDER / 2 - TICKSIZE));

      /* draw top tick marks */
      XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
           (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions),
            (int)(GRAPHBORDER / 2),
           (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions),
            (int)(GRAPHBORDER / 2 + TICKSIZE));

    }

    sprintf(buffer,"%.4G", (maxX - minX) * i / xdivisions + minX);
    XDrawString(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
          (int)(lmargin + GRAPHBORDER / 2 + (maxX - minX) * xscale * i / xdivisions -
XTextWidth(w->lgraph.font,buffer,strlen(buffer)) / 2),
          height - 1,
          buffer,
          strlen(buffer));
  }

  /* Draw Major Y Tick Marks */
  for (i = 0; i <= ydivisions; i++)
  {
    if (i != 0 && i != ydivisions)
    {
      if (w->lgraph.ydividers)
      {
        XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.grid_GC,
              (int)(lmargin + GRAPHBORDER / 2),
             (int)(height - i * (maxY - minY) * yscale / ydivisions - GRAPHBORDER / 2),
          (int)(lmargin + width - GRAPHBORDER / 2),
         (int)(height - i * (maxY - minY) * yscale / ydivisions - GRAPHBORDER / 2));
    }

    /* Draw Y axis tick marks */
    XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
        (int)(lmargin + GRAPHBORDER / 2 + fabs(minX) * xscale + TICKSIZE / 2),
        (int)(height - i * (maxY - minY) * yscale / ydivisions - GRAPHBORDER / 2),
        (int)(lmargin + GRAPHBORDER / 2 + fabs(minX) * xscale - TICKSIZE / 2),
        (int)(height - i * (maxY - minY) * yscale / ydivisions - GRAPHBORDER / 2));

    /* Draw left tick marks */
    XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
        (int)(lmargin + GRAPHBORDER / 2),
        (int)(height - i * (maxY - minY) * yscale / ydivisions - GRAPHBORDER / 2),
        (int)(lmargin + GRAPHBORDER / 2 + TICKSIZE),
        (int)(height - i * (maxY - minY) * yscale / ydivisions - GRAPHBORDER / 2));

    /* Draw right tick marks */
    XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
        (int)(lmargin + width - GRAPHBORDER / 2),
        (int)(height - i * (maxY - minY) * yscale / ydivisions - GRAPHBORDER / 2),
        (int)(lmargin + width - GRAPHBORDER / 2 - TICKSIZE),
        (int)(height - i * (maxY - minY) * yscale / ydivisions - GRAPHBORDER / 2));
   }

   sprintf(buffer,"%.4G", (maxY - minY) * i / ydivisions + minY);
   XDrawString(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,labelwidth +
LABELSEPARATION,
        (int)(height - i * (maxY - minY) * yscale / ydivisions - GRAPHBORDER / 2 + w-
>lgraph.font->ascent / 2),
        buffer,
        strlen(buffer));
  }

  /* Draw Y Axis */
  XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
      (int)(lmargin + GRAPHBORDER / 2 + fabs(minX) * xscale),
      (int)(GRAPHBORDER / 2),
      (int)(lmargin + GRAPHBORDER / 2 + fabs(minX) * xscale),
      (int)(height - GRAPHBORDER / 2));

  /* Draw Y Axis Label */
  XDrawString(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,0,
      height / 2,
        w->lgraph.ylabel,
        strlen(w->lgraph.ylabel));

  /* Draw X Axis */
  XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
        (int)(lmargin + GRAPHBORDER / 2),
        (int)(fabs(maxY) * yscale + GRAPHBORDER / 2),
        (int)(lmargin + width - GRAPHBORDER / 2),
        (int)(fabs(maxY) * yscale + GRAPHBORDER / 2));

  /* Draw the X Axis Label */
  XDrawString(XtDisplay(w),XtWindow(w),w->lgraph.normal_GC,
        w->core.width / 2 - XTextWidth(w->lgraph.font,w->lgraph.xlabel,strlen(w-
>lgraph.xlabel)) / 2,
        w->core.height - w->lgraph.font->descent,
        w->lgraph.xlabel,
        strlen(w->lgraph.xlabel));

  /* Draw Data */
  for (data = w->lgraph.ldata; data && data->nextdata; data = data->nextdata)
  {
    for (i = 0; i < data->yvc; i++)
    {
      x1 = lmargin + xscale * (data->x - minX) + GRAPHBORDER / 2;
      x2 = lmargin + xscale * (data->nextdata->x - minX) + GRAPHBORDER / 2;
      y1 = height - yscale * (data->yv[i] - minY) - GRAPHBORDER / 2;
      y2 = height - yscale * (data->nextdata->yv[i] - minY) - GRAPHBORDER / 2;

/* printf("XS:%f YS:%f X1:%d Y1:%d X2:%d Y2:%d\n",xscale,yscale,x1,y1,x2,y2); */ /
* debugging only */
      /* draw only if something will be actually drawn */
      XDrawLine(XtDisplay(w),XtWindow(w),w->lgraph.color_GC[i%8],x1,y1,x2,y2);
    }
  }
}

lgraphFree(w)
LgraphWidget w;
{
  LDATA *datarec;

  for (datarec = w->lgraph.ldata; datarec; datarec = datarec->nextdata)
  {
    free(datarec->yv);
    free(datarec);
    }
  }
```

```
/*
  Image.h

  Change History
      3-Nov-90/mwl - Original Issue
*/

#ifndef _XImage_h
#define _XImage_h

/************************************************************************
 *
 * Image Widget
 *
 ************************************************************************/

#include <X11/Core.h>

/* Resources:

 Name    ClassRepTypeDefault Value
 ----    -------------------------
 bitmap    PixmapPixmapNone
 lowcolor        Lowcolor        String      Blue
 highcolor       Highcolor       String      Red
 numcolors        Numcolors       Int        20
 colormap         Colormap        String      None

*/

/*  Private Atoms */

extern WidgetClass     imageWidgetClass;

typedef struct _ImageClassRec   *ImageWidgetClass;
typedef struct _ImageRec      *ImageWidget;

#define XtNlowcolor "lowcolor"
#define XtCLowcolor "Lowcolor"
#define XtNhighcolor "highcolor"
#define XtCHighcolor "Highcolor"
#define XtNnumcolors "numcolors"
#define XtCNumcolors "Numcolors"
#define XtNcolormap "colormap"
#define XtCColormap "Colormap"
```

```
#endif /* _XImage_h */
```

```
/*
  ImageP.h - Private definitions for Image widget

  Change History
      3-Nov-90/mwl - Original Issue
 */

#ifndef _XImageP_h
#define _XImageP_h

#include "Image.h"
#include <X11/CoreP.h>

/*********************************************************************
 *
 * Lgraph Widget Private Data
 *
 *********************************************************************/

typedef struct LDATA_TAG
{
struct LDATA_TAG *nextdata;
double x;
double *yv;
int yvc;
} LDATA;


/***********************************
 *
 *  Class structure
 *
 ***********************************/


  /* New fields for the Lgraph widget class record */
typedef struct _ImageClass
 {
   int makes_compiler_happy;  /* not used */
  } ImageClassPart;

  /* Full class record declaration */
typedef struct _LgraphClassRec {
   CoreClassPartcore_class;
   ImageClassPart    image_class;
} ImageClassRec;

extern ImageClassRec imageClassRec;

/**************************************
 *
 * Instance (widget) structure
 *
 **************************************/

  /* New fields for the Image widget record */

typedef struct {
  /* resources */
  XtCallbackList callbacks;
  Pixel           foreground;
  Pixmap          current_pixmap;
  char            *lowcolor;
  char            *highcolor;
  char            *colormap;
  int             numcolors;

  /* Private areas */
  GC              normal_GC;
  Pixel           colors[128];
  XImage          *current_image;
  int             depth;
  int             isrealized;
} ImagePart;


/* Full widget declaration */
typedef struct _ImageRec {
  CorePart      core;
  ImagePart     image;
} ImageRec;

#endif /* _XImageP_h */
```

```
/*
  Image.c

  A Widget used to smoothly display pixmaps.

  Change History:
       12-Sep-90/mwl - Original Issue
        3-Nov-90/mwl - Released

*/

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <math.h>
#include <X11/Xlib.h>
#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include "ImageP.h"

/**************************************************************
 *
 * Full class record constant
 *
 **************************************************************/

/* Private Data */

static char defaultTranslations[] =
    "<Btn1Down>:notify()";

#define offset(field) XtOffset(ImageWidget, field)

static XtResource resources[] = {
  {XtNcallback, XtCCallback, XtRCallback, sizeof(XtPointer),
    offset(image.callbacks), XtRCallback, (XtPointer)NULL},
  {XtNbitmap,XtCPixmap,XtRBitmap,sizeof(Pixmap),
    offset(image.current_pixmap),XtRImmediate, (caddr_t)None },
  {XtNforeground,XtCForeground,XtRPixel,sizeof(Pixel),
    offset(image.foreground),XtRString, "XtDefaultForeground" },
  {XtNlowcolor,XtCLowcolor,XtRString, sizeof (char *),
    offset(image.lowcolor),XtRString, "blue" },
  {XtNhighcolor,XtCHighcolor,XtRString, sizeof (char *),
    offset(image.highcolor),XtRString, "red" },
  {XtNnumcolors, XtCNumcolors, XtRInt, sizeof (int),
    offset(image.numcolors), XtRImmediate, (caddr_t)20 },
  {XtNcolormap, XtCColormap, XtRString, sizeof (char *),
    offset(image.colormap), XtRString, "" }
};
#undef offset

static Boolean SetValues();
static void Initialize(), Redisplay(), Notify();
static void Destroy();
static void ClassInitialize();

static XtActionsRec actionsList[] =
{
 {"notify",Notify}
};

#define SuperClass ((CoreWidgetClass)&coreClassRec)

ImageClassRec imageClassRec = {
  {
   (WidgetClass) SuperClass,/* superclass */
   "Image",/* class_name */
   sizeof(ImageRec),/* size */
   NULL,/* class_initialize */
   NULL,/* class_part_initialize */
   FALSE,/* class_inited */
   Initialize,/* initialize */
   NULL,/* initialize_hook */
   XtInheritRealize,/* realize */
   actionsList,/* actions */
   XtNumber(actionsList),/* num_actions */
   resources,/* resources */
   XtNumber(resources),/* resource_count */
   NULLQUARK,/* xrm_class */
   FALSE,/* compress_motion */
   TRUE,/* compress_exposure */
   TRUE,/* compress_enterleave   */
   FALSE,/* visible_interest */
   Destroy,/* destroy */
   XtInheritResize,/* resize */
   Redisplay,/* expose */
   SetValues,/* set_values */
   NULL,/* set_values_hook */
   XtInheritSetValuesAlmost,/* set_values_almost */
   NULL,/* get_values_hook */
  NULL,/* accept_focus */
  XtVersion,/* version */
  NULL,/* callback_private */
  defaultTranslations,/* tm_table */
  XtInheritQueryGeometry,/* query_geometry */
  XtInheritDisplayAccelerator,/* display_accelerator */
  NULL/* extension */
 }, /* CoreClass fields initialization */
 {
  0,                    /* field not used   */
 }, /* ImageClass fields initialization */
};

 /* for public consumption */
WidgetClass imageWidgetClass = (WidgetClass) &imageClassRec;

/*****************************************************************
 *
 * Private Procedures
 *
 *****************************************************************/

static GC
Get_GC(cbw, fg, bg)
ImageWidget cbw;
Pixel fg, bg;
{
 XGCValuesvalues;

 values.foreground  = fg;
 values.background= bg;
 values.cap_style = CapProjecting;
 values.line_width  = 0;

 return XtGetGC((Widget)cbw,
 (GCForeground|GCBackground|GCLineWidth|GCCapStyle),
 &values);
}


/* ARGSUSED */
static void
Initialize(request, new, args, num_args)
Widget request, new;
ArgList args;/* unused */
```

```
Cardinal *num_args;/* unused */
{
  ImageWidget cbw = (ImageWidget) new;
  Window root;
  int x,y,w,h,bw,depth,displaydepth;
  XGCValues values;
  int i;
  int red, green, blue;
  int incred,incgreen,incblue;
  Colormap cmap;
  XColor lowdef,highdef;
  XColor def;

  displaydepth = DefaultDepthOfScreen(XtScreen(cbw));
  cbw->image.depth = displaydepth;

  if (cbw->image.current_pixmap != NULL)
  {
    XGetGeometry(XtDisplay(cbw),cbw->image.current_pixmap,&root,&x,&y,&w,&h,&-
bw,&depth);
    cbw->core.width = w;
    cbw->core.height = h;
  }
  else if (!cbw->core.width || !cbw->core.height)
  {
fprintf(stderr,"Image Widget:  Setting width and height manually\n");
    cbw->core.width = 400;
    cbw->core.height = 400;
  }

  if (displaydepth >= 8)
  {
    cmap = DefaultColormapOfScreen(XtScreen(cbw));

    if (!XParseColor(XtDisplay(cbw),cmap,cbw->image.lowcolor, &lowdef))
      printf("Image Widget Error:  Can't find low color definition\n");

    if (!XParseColor(XtDisplay(cbw),cmap,cbw->image.highcolor, &highdef))
      printf("Image Widget Error:  Can't find high color definition\n");

    red = lowdef.red;
    blue = lowdef.blue;
    green = lowdef.green;

    incred = ((int)highdef.red - (int)lowdef.red) / cbw->image.numcolors;
    incgreen = ((int)highdef.green - (int)lowdef.green) / cbw->image.numcolors;
    incblue = ((int)highdef.blue - (int)lowdef.blue) / cbw->image.numcolors;

    for (i = 0; i < cbw->image.numcolors;)
    {
      def.red = red;
      def.green = green;
      def.blue = blue;

      if (!XAllocColor(XtDisplay(cbw),cmap,&def))
      {
        printf("Image Widget Error:  Can't allocate colorcell\n");
        cbw->image.colors[i] = cbw->image.foreground;
      }
      else
        cbw->image.colors[i] = def.pixel;

      i++;

      red += incred;
      green += incgreen;
      blue += incblue;
    }
  }
  else  /* black and white display, allocates only two colors */
  {
    cbw->image.colors[0] = cbw->core.background_pixel;
    cbw->image.colors[1] = cbw->image.foreground;
    cbw->image.numcolors = 2;
  }

  /* get GC so we can draw */
  values.foreground = cbw->image.foreground;;
  values.background = cbw->core.background_pixel;
  cbw->image.normal_GC = XCreateGC(XtDisplay(cbw),RootWin-
dowOfScreen(XtScreen(cbw)),GCForeground|GCBackground,&values);

  /* general initialization */
  cbw->image.current_image = (XImage *)NULL;
  cbw->image.isrealized = 0;
}


/***************************
*
```

```
* Action Procedures
*
***************************/

/* ARGSUSED */
static void
Notify(w,event,params,num_params)
Widget w;
XEvent *event;
String *params;/* unused */
Cardinal *num_params;/* unused */
{
 ImageWidget cbw = (ImageWidget)w;

 XtCallCallbacks(w,XtNcallback,(caddr_t)event);
}

/*
 * Repaint the widget window
 */

static void Redisplay(w, event, region)
ImageWidget w;
XExposeEvent *event;
Region region;
{
  w->image.isrealized = 1;

  if (w->image.current_pixmap)
    XCopyArea(XtDisplay(w),w->image.current_pixmap,XtWindow(w),
          w->image.normal_GC,0,0,w->core.width,w->core.height,0,0);
  else if (w->image.current_image)
    XPutImage(XtDisplay(w),XtWindow(w), w->image.normal_GC, w->image.current_i-
mage,
          0, 0, 0, 0, w->core.width, w->core.height);
}

/*
 * Destroy Method
 */
static void Destroy(w)
ImageWidget w;
{
  Colormap cmap;
  if (w->image.current_pixmap)
    XFreePixmap(XtDisplay(w),w->image.current_pixmap);

  if (w->image.current_image)
    XDestroyImage(w->image.current_image);

  XFreeGC( XtDisplay(w), w->image.normal_GC);
}

/*
 * Set specified arguments into widget
 */

/* ARGSUSED */
static Boolean
SetValues (current, request, new)
Widget current, request, new;
{
  ImageWidget oldcbw = (ImageWidget) current;
  ImageWidget cbw = (ImageWidget) new;
  Boolean redisplay = False;
  XGCValues values;

  if (oldcbw->image.current_pixmap != cbw->image.current_pixmap)
    redisplay = True;

  if ( (oldcbw->image.foreground!= cbw->image.foreground) ||
      (oldcbw->core.background_pixel != cbw->core.background_pixel))
  {
    XFreeGC(XtDisplay(new), cbw->image.normal_GC);

   values.foreground = cbw->image.foreground;
   values.background = cbw->core.background_pixel;
   cbw->image.normal_GC = XCreateGC(XtDisplay(cbw),XtWin-
dow(cbw),GCForeground|GCBackground,&values);

    redisplay = True;
  }

  return (redisplay);
}


/*
 * Private commands
 */
void ImagePixmapSet(w,map)
ImageWidget w;
Pixmap map;
{
  int x,y,width,height,bw,depth;
  Window root;
  Arg list[3];

  /* look at the size of the new pixmap.  Resize the window if necessary */
  XGetGeometry(XtDisplay(w),map,&root,&x,&y,&width,&height,&bw,&depth);
  if (width > w->core.width || height > w->core.height)
  {
    /* XtResizeWidget(w,width,height,w->core.border_width); */
    XtMakeResizeRequest(w,width,height,&width,&height);
  }

  /* set the new pixmap to the current one, and blit it to the screen */
  w->image.current_pixmap = map;

  if (!w->image.isrealized)
    return;

  XCopyArea(XtDisplay(w),w->image.current_pixmap,XtWindow(w),w->image.nor-
mal_GC,0,0,w->core.width,w->core.height,0,0);
}

void ImageTest(w)
ImageWidget w;
{
  int i,j;

  if (!w->image.isrealized)
    return;

  for (i = 0; i < w->image.numcolors; i++)
  {
    XSetForeground(XtDisplay(w),w->image.normal_GC,w->image.colors[i]);
    for (j = 0; j <= w->core.height / w->image.numcolors; j++)
    {
      XDrawLine(XtDisplay(w),XtWindow(w),w->image.normal_GC,
        0,i * w->core.height / w->image.numcolors + j,
        w->core.width,i * w->core.height / w->image.numcolors + j);
    }
  }
```

```
}

void ImageScientificSet(w,data)
ImageWidget w;
char data[];
{
  register int x,y,position;
  Pixel color;
  int idata;
  char *buffer;

  if (!w->image.current_image)  /* if no image has yet been allocated, allocate one for an 8-
bit display */
  {
    buffer = malloc(w->core.width * w->core.height);
    if (!buffer)
    {
     fprintf(stderr,"Error:  Can't Allocate Image Data Area [%d] %d %d\n",w->core.width
* w->core.height,w->core.width,w->core.height);
      return;
    }

    w->image.current_image = XCreateImage(XtDisplay(w),
                              DefaultVisualOfScreen(XtScreen(w)),
                              w->image.depth,ZPixmap, 0, buffer,
                              w->core.width, w->core.height,
                              8, 0);

    if (!w->image.current_image)
      fprintf(stderr,"Error:  Can't create image\n");
  }

  if (w->image.depth > 1)  /* color display */
  {
    position = 0;
    for (y = 0; y < w->core.height; y++)
      for (x = 0; x < w->core.width; x++)
      {
        idata = data[position++] % w->image.numcolors;
        color = w->image.colors[idata];

        XPutPixel(w->image.current_image,x,y,color);
      }
  }
  else  /* mono display */
  {
    position = 0;
    for (y = 0; y < w->core.height; y++)
      for (x = 0; x < w->core.width; x++)
      {
        color = ((data[position++]) ?
              w->image.colors[1] : w->image.colors[0]);
        XPutPixel(w->image.current_image,x,y,color);
      }
  }

  if (!w->image.isrealized)
  {
    return;
  }

  XPutImage(XtDisplay(w),XtWindow(w),w->image.normal_GC,w->image.current_im-
age,
        0,0,0,0,w->core.width,w->core.height);
}

void ImageSet(w,image)
ImageWidget w;
XImage *image;
{
  /* change to the new image */
  w->image.current_image = image;

  if (!w->image.isrealized)
    return;

  /* blit it to the display */
  XPutImage(XtDisplay(w),XtWindow(w),w->image.normal_GC,w->image.current_im-
age,
        0,0,0,0,w->core.width,w->core.height);

}
void ImageDeltaSet(w,data)
ImageWidget w;
unsigned char *data;
{
  int done = 0;
  int bytes,i;
  int x,x2,y,y2;
  unsigned int position = 0;
```

```
Pixel color;

/* if there isn't a preallocated image, a delta isn't possible */
if (!w->image.current_image)
{
  utilError("Error:  No current image for delta processor");
  return;
}

while (!done)
{
  switch(*data)  /* depending on command */
  {
    case 'S':
      position += *(data + 1);
      data += 2;
      break;

    case 'U':
      bytes = *(data + 1);
      data += 2;

      if (w->image.depth > 1)
        for (i = 0; i < bytes; i++, position++, data++)
        {
          if (*(data) >= w->image.numcolors)
            color = w->image.colors[w->image.numcolors - 1];
          else
            color = w->image.colors[*(data)];
          XPutPixel(w->image.current_image,position % w->core.width,
                position / w->core.width,color);
        }
      else
        for (i = 0; i < bytes; i++, position++, data++)
        {
          color = w->image.colors[((*(data) >= 1) ? 1 : 0)];
          XPutPixel(w->image.current_image,position % w->core.width,
                position / w->core.width,color);
        }

      break;

    case 'D':
      done = 1;
      break;
      default:
        fprintf("Image Widget Error:  Invalid Delta Command [%c]\n",*data);
        data++;
        break;
    }
  }

  if (!w->image.isrealized)
  {
    utilError("Error:  Trying to delta to an unrealized widget");
    return;
  }

  XPutImage(XtDisplay(w),XtWindow(w),w->image.normal_GC,w->image.current_im-
age,
         0,0,0,0,w->core.width,w->core.height);
}
```

VITA

Mark Wayne Lenox received his B.S.E. in Systems Engineering from Arizona State University in 1989. He worked as a programmer for Quincy Street, Corporation in Phoenix Arizona for several years before going to work for Cray Research, Inc., in 1989. With the realization that a B.S. just wasn't enough, he moved to College Station Texas to pursue an M.S in Electrical Engineering. While studying at Texas A&M he worked at the Texas A&M Supercomputer Center in the areas of scientific visualization, distributed processing, and UNIX system programming. In his spare time, he competed with the A&M water ski team, and currently holds the A&M mens trickskiing record. After graduation, Mark plans to live, work, and ski (although not necessarily in that order) in the Dallas, Texas area.

Further information about SP is available from the Texas A&M Supercomputer Center, 006 WERC, College Station, TX 77843.