Lawrence G. Palmer
Ricky S. Palmer

Shared Desktop: A Collaborative Tool for Sharing 3-D Applications among Different Window Systems

The DIGITAL Workstations Group has designed a software application that supports the sharing of three-dimensional graphics and audio across the network. The Shared Desktop application enables the collaborative use of any application over local and long-distance networks, as well as interoperation among Windows- and UNIX-based computers. Its simple user interface employs screen capture and data compression techniques and a high-level protocol to transmit packets using TCP/IP over the Internet.

An advanced product development effort undertaken by graphics engineers in the DIGITAL Workstations Group led to the creation of a new software application called Shared Desktop. One project goal was to enable collaboration among users of three-dimensional (3-D) graphics workstations that run either the UNIX or the Windows NT operating system. Another goal was to allow these users to access the high-performance 3-D capabilities of their office workstations from their laptop computers or home-based personal computers (PCs) that run the Windows 95 system and do not have 3-D graphics hardware. This goal necessitated a cross–operating-system application that could efficiently and effectively handle 3-D graphics in real time and share these graphics with machines such as laptop computers and PCs.

In this paper, we begin with a discussion of the software currently available for computer collaboration. We then discuss the development of the Shared Desktop application, focusing on the user interface, protocol splitting, screen capture and data handling, and dissimilar frame buffers. We conclude with sections on additional uses and future directions of the Shared Desktop product.

Current Collaboration Software

Computer collaboration may be defined as the interaction between computers and their human users over a local or long-distance network. In general, it involves a transfer of textual, graphical, audible, and visual information from one collaborator to another. The participants share control information either by means of computer-generated synchronization events or by human voice and visual movement.[1]

Specifically, computer collaboration involves communicating and sharing data among participants who can be located anywhere in a building, a city, a country, or the world. Each participant has either a PC, a workstation, or a laptop computer. Some machines contain 3-D graphics adapters with hardware acceleration. (Computer-aided design/computer-assisted manufacturing [CAD/CAM] applications like Parametric Technology Corporation's [PTC] Pro/ENGINEER use hardware accelerators through OpenGL[2] or

Direct3D[3] programming protocols.) Other computers do not contain 3-D accelerator boards and provide 3-D capabilities through software-only routines on two-dimensional (2-D) hardware. In a typical collaboration, a person wanting to share a specific 3-D graphical display of a part or model telephones others to discuss the design in progress. After the initial contact, the collaborators may continue the telephone call or switch to the audio function of the application. The graphics part appears on each participant's screen along with associated keyboard and mouse events. As the collaborators discuss the work, they may each interact with the display to highlight, rotate, and change the look or design of the 3-D part. In this way, even though the participants are separated by some distance, they may interact as if they were all sitting around a table working, conversing, and designing the 3-D part.

Current software that facilitates computer-based collaboration runs through a range of capabilities from the earliest forms of electronic mail to the most recent offerings of complete collaborative sharing of the computer. Examples include WinFrame technology from Citrix Systems, Inc., NetMeeting from Microsoft Corporation, Netscape Communicator from Netscape Communications Corporation, and other products from Sun Microsystems, Hewlett-Packard, and Silicon Graphics Inc. These packages offer levels of computer sharing and collaboration from videoconferencing and file sharing to full application sharing. Each implementation runs on specific operating systems. Although they use various underlying communication protocols, most recent designs work over local area and wide area networks (LANs/WANs), including the Internet. For example, the NetMeeting product provides conferencing tools like chat, whiteboard, file transfer, audio and videoconferencing, and non-real-time, selected-window 2-D application sharing over T120 protocols layered on the Transmission Control Protocol/Internet Protocol (TCP/IP).[4] NetMeeting runs only on Microsoft platforms (Windows 95 and Windows NT operating systems). The current products are deficient, however, in that they do not support multiple operating systems, do not operate in real time, and do not share 3-D graphics.

User Interface

In this section, we describe our choice of a simple user interface for the sharing area of a desktop and our design of the Shared Desktop Manager for client-server computing.

Many collaboration tools for sharing computer information (graphical desktop, keyboard, mouse, and audio of a given computer) were complete systems and required too much effort on the part of the users just to learn how to share information. A focus on learning collaboration tools often requires users to become experts in the collaboration software rather than in the applications that they may share. Since the various 3-D graphics packages that needed to be shared were complicated in themselves, we decided to implement a simple user interface in the Shared Desktop application that nearly all audiences could easily learn and use.

In the Shared Desktop design, we designated part of the desktop screen as a sharing area. Graphics objects such as icons and applications located within the sharing area can be accessed by all conference participants. To share a new application, a participant moves the application into the sharing area. To remove an application, a participant moves it outside the sharing area. If the sharing area encompasses the entire desktop of the initiating participant, all applications are shared. We used standard pull-down menus and widgets provided by either the UNIX X Motif toolkit or the Microsoft Windows libraries. We named the sharing area the "viewport"; it is viewed on the desktop as a user-defined area of rectangular size and location. Any graphical object placed into the viewport is marked as shareable with client users in a collaboration. We designed the viewport so that it is always on the bottom of a given stack of windows on a desktop. Thus, when Shared Desktop is minimized, so is its viewport. The objects that had been within the viewport are returned to the initiator's desktop and are no longer shared. With a quick minimization, the server collaborator can pause any sharing that was in progress without disconnecting from the client users.

Figure 1 illustrates a UNIX server with a Shared Desktop viewport connected to several client systems. The server's viewport contains no shared objects within its confines, and each client screen shows a viewport received from the server.

The viewport can be set to represent the entire visible desktop, or it can be set to equal only the size of a given application on the screen. Accordingly, a user who is acting as the server can determine how much of a given desktop to share among the client collaborators. The concept of a viewport is valuable because the principal collaborator (at the server) can quickly glance at the screen and determine what to capture and send to other participants. (The objects and applications sent from the server are designated by solid lines in Figure 1.) The Shared Desktop application requires no further action to set up an application for sharing.

Each client sends keyboard and mouse events to the server to control any application present in the viewport (remote control is shown as dashed lines in Figure 1). Server and clients synchronize cursor movements so that any conference member can watch as others make changes to a shared application. This allows the cursor to become a pointer during a session. Shared Desktop implements an "anarchy" form of remote control, with all mice and keyboards active simultaneously.

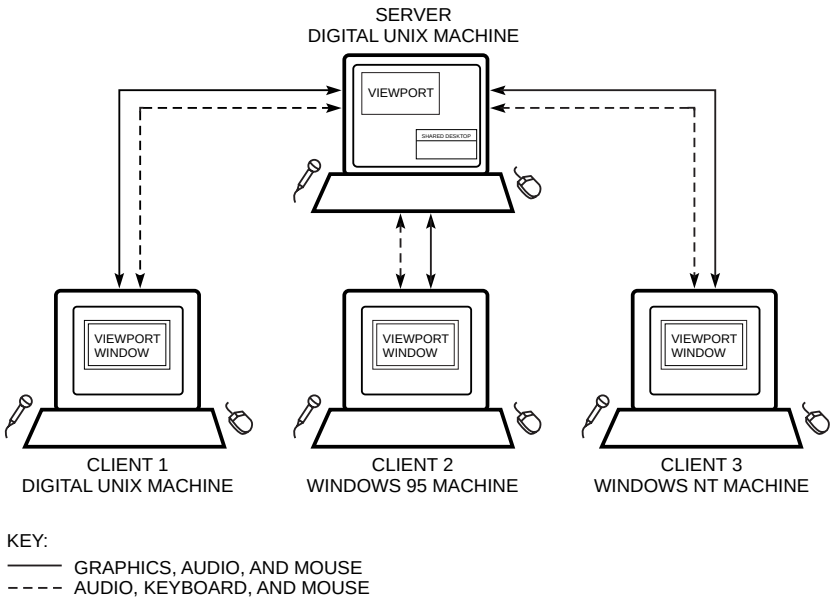

Figure 1
Server Desktop with Viewport and Clients

When a user initiates a collaboration, the audio is off by default but remains integral to a session as a convenience (as opposed to using the telephone). Through a pull-down menu operation, the server enables audio for all participants in one operation. The usual audio management tools used to set microphone recording levels and speaker/headset play-back levels are available. As Figure 1 indicates, the UNIX machine collects audio and distributes it to the three client collaborators. Likewise, the three clients collect audio and send it back to the server for mixing. In this way, all participants can hear one another and interact with whatever objects appear in the viewport on the server's screen.

Figure 2 shows the Shared Desktop Manager from the initiator's viewport running on the UNIX server. A participant may use a Session pull-down menu to control the viewport and to connect and disconnect other conference members. The Options menu allows for audio, remote cursor, and call-back control. The application's Help pull-down menu provides the usual help information similar to a Windows help facility or a Web browser's help. The window lists the status of attached clients.

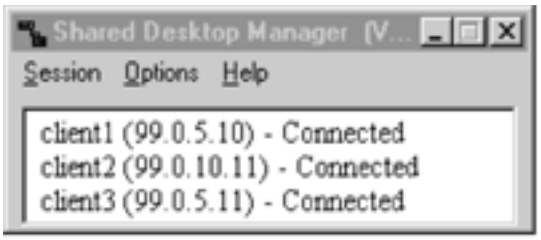

Figure 2
Shared Desktop Manager

Upon connection, participants can hear and interact with the server. The resultant audio dialogue combined with the graphics, keyboard, and mouse interactions facilitate a collaboration environment in which participants share an application. Since each user can operate a separate mouse and keyboard, the audio channel acts as a synchronization mechanism to indicate which collaborator controls the shared applications at any given moment. The participants communicate their actions verbally, interacting in much the same way as people who are sitting around a table and working.

Design Features

For our implementation, we concentrated on three principal areas: protocol splitting, screen capture and data handling, and dissimilar frame buffers. In this section, we discuss our investigation into using a protocol splitter and our decision to rely on screen capture and data handling. We also discuss dissimilar frame buffers.

Protocol Splitting

We looked for a way to distribute 3-D graphics among workstations and PCs that would be independent of the application, graphics protocol, architecture, operating system, and windowing system. On UNIX, we found application sharing provided by distributed windows protocols. For example, the X Protocol[5] allows a user to send an application to a nonlocal display and to send X applications protocol messages to several screens simultaneously. A protocol splitter, however, has disadvantages due to its requirements for bandwidth, programming, and latency.

Protocol splitters require distribution of graphics commands and display lists by means of a network. Three-dimensional models often contain megabytes of graphical information that describe specific screen operations. When displaying a model locally, these graphics operations move quickly and easily over system buses that are capable of handling hundreds of megabytes per second. However, when these same graphics objects are copied over computer networks, the amount of information can overload even the highest-speed networks. For example, using a 100-megabyte (MB) Pro/ENGINEER truck assembly, a current generation 3-D workstation can load, display, and rotate the truck once in approximately 2 minutes. The same operation between two identical 3-D workstations takes 20 minutes when performed by a distributed protocol, and the rotation of the truck does not appear fluid to the user. If the same data or application is duplicated on every machine, only updates with synchronizing events are distributed, but this requires that all machines have the same graphics hardware.

The programming software needed for interoperation among dissimilar operating and windowing systems using protocol splitting is quite involved. The ability to support X11 desktops, Windows 95 desktops, and Windows NT desktops while using multiple 3-D protocols like OpenGL and Direct3D would require that these protocols exist on all platforms.

Latency requirements for 3-D are very stringent. Thus, any network jitter makes even the best network link create breakup (visual distortions) when rotating 3-D objects. Network jitter also causes delays in sending window protocol messages; as a delay increases, the window events may no longer be useful. For example, when rotating a 3-D object, the delayed events must propagate as the network permits although this may once again congest the network since the events may no longer be needed. The object has now rotated to a new view. The ability to drop some protocol messages in a time-critical way is a requirement for collaborating with 3-D objects, and the protocol splitter approach to sharing has no solution for this problem.

Screen Capture and Data Handling

To overcome these issues, we investigated capturing the screen display, the final bitmap result of the interaction of graphics hardware and software that the viewer sees. Capturing the screen is in itself nothing new; it has been used for some time to include screen visuals in document preparation. Initially, we were skeptical that capturing the screen display could be a useful mechanism since the amount of data on a screen can be prodigious. Screen graphics depth and resolution can make the amount of data in any given graphics object very large. For example, for a 24-plane frame buffer with a 1,280 by 1,024 resolution, the total amount of data to capture would be (24 x 1,280 x 1,024)/8 or about 4 MB. Using the computational power of the Alpha microprocessor for reducing the data, we continued our investigation. We found that this approach requires the windowing system to perform screen capture by means of a non-CPU-intensive routine (direct memory access [DMA] as opposed to programmed I/O). Based on our tests, we concluded that screen capture technology would be easier to implement than a protocol splitter, would have better latency for 3-D operations than a protocol splitter, and would be easily adaptable to the various windowing systems and 3-D protocols we wished to have interoperate.

Graphics Compression The screen capture approach requires a number of steps to efficiently prepare the data for transmission. First, the contents of a viewport are captured, and the sample is saved for comparison with successive samples. Second, the captured viewport samples are differenced to find screen pixels that have not been changed and delta values for those that have been changed. Third, the resultant array of values is compressed by a fast, run-length encoding (RLE) of the array of difference samples. A more CPU-intensive compression may now be applied. The fourth step is to apply LZ77 compression that reduces the remaining RLE data to its smallest form. In step four, the original data has been reduced while retaining its characteristics so that it can be restored (uncompressed) without loss on a receiving computer. This final lossless stage of compression occurs only if it reduces the amount of data and if the network was busy during a previous transmission. Lossless compression is important for the nondestructive transfer of data from the server's screen to the clients' screens and has application in industry. As an example, consider a doctor who is sharing an x-ray with an out-of-town colleague. If the graphics were compromised by a lossy compressor, the collaborators could not be guaranteed that the transmitted x-ray was identical to the one sent. With the Shared Desktop application, the doctor who is sending the x-ray is guaranteed that the original graphics are restored on the colleague's display. In some forms of compression, data is thrown out by the algorithm and never restored, so that the final screen data may not accurately reflect the original graphics. Figure 3 shows the steps in the capture and compression sequence.

On the Alpha architecture, these compression steps are performed as 64-bit operations, both in the data manipulation and the compression algorithms. The Alpha architecture lends itself to a fast and efficient implementation of the algorithms, so that the capture of the viewport and the multistage compression of the data can be accomplished in real time. Approximately half of the number of instructions is used on a processor that is twice as fast as a 32-bit architecture. In addition to its 64-bit routine, the RLE is implemented as a 32-bit routine and as a comparison routine.

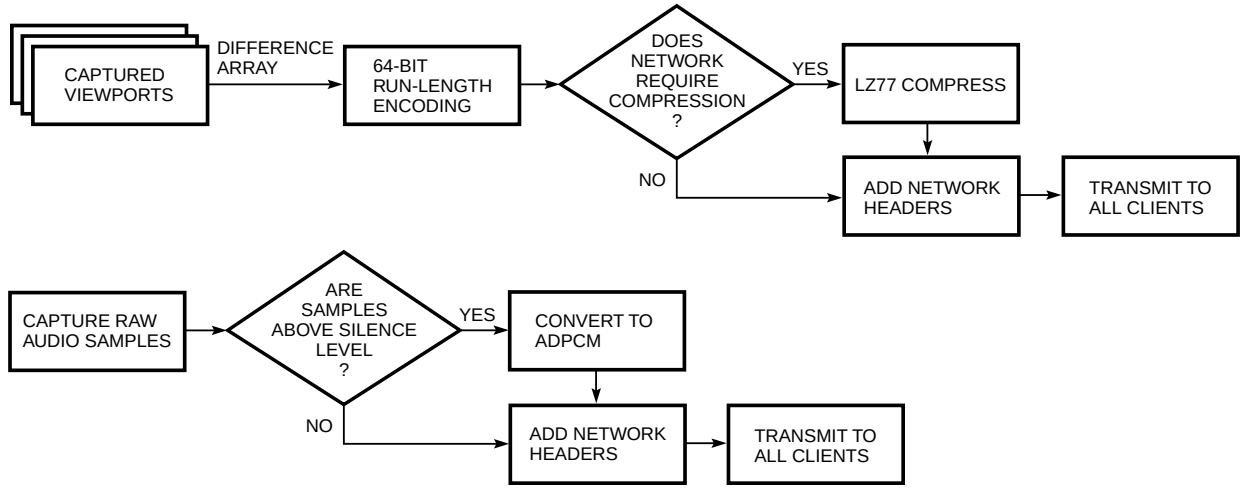

Figure 3
Graphics and Audio Compression Data Flow Diagram

Audio Compression Similar to the graphics compression described, the audio compression in Shared Desktop involves several steps. First, the audio samples are captured through a microphone and sound card combination. These samples are compared with the background noise level (determined prior to beginning a conference) to see if the samples are useful. Samples below the background noise level are not transferred. This implements a silence detection method whereby only useful samples will advance to the next level of compression. Second, the next compression uses G.711 or other similar audio compression standards and converts adaptive differential pulse code modulation (ADPCM) samples at 64 kilobits per second into 16 kilobits per second (4:1 lossy compression).[6] Third, this data is then ready for transfer to a receiving computer so that it may be decompressed and output to a speaker or a headset. The audio stream resulting from these steps generates at most 16 kilobits per second when someone is speaking, and no output when it is silent. Figure 3 also shows the audio compression steps.

Data Transmission After the graphics and audio data are collected and compressed, they are combined and transmitted across the network by a patented, higher-level protocol that ensures timely delivery of each packet.[7] All packets are sent using TCP/IP over the Internet. Although the higher-level protocol does not guarantee true real-time characteristics, the patented protocol allows for coherent audio, synchronization of graphics and cursor events, and near real-time graphics animation.

As an example, the screen capture shown in Figure 4 displays a 100-MB Pro/ENGINEER assembly being shared through the Shared Desktop application. The Shared Desktop Manager system (system where the assembly database resides) is an AlphaStation 500 workstation running the DIGITAL UNIX operating system with a PowerStorm 4D60 graphics controller. In this example, an 800- by 600-pixel by 24-bit Shared Desktop viewport is being captured, compressed, and transmitted to the Shared Desktop client system at about five updates per second. The update rate is determined by the capture viewport size, the extent of detail changes between captures, the amount of processing power needed by the application to make changes to the model, and the speed of the network. In this example, when rotating the truck assembly, a compressed stream of 400 to 500 kilobits per second is generated and represents the five updates per second mentioned. A simple assembly might be able to do a rotation with Shared Desktop capturing and transmitting 15 updates per second, and a more complicated model (like the truck assembly shown) would receive fewer updates per second.

Dissimilar Frame Buffers

To complete the requirements of our implementation, we needed to share graphics information across dissimilar hardware, i.e., machines with different graphic frame buffer depths. The frame buffer depth refers to the amount of storage the graphics adapter gives to each displayed pixel on the screen. A 16-bit-deep display assigns each pixel a 16-bit value to represent the pixel. This representation is usually the color information for the pixel, i.e., what color the user sees for a given pixel.[8] The frame buffer depths are a necessary reality since different graphics devices have widely varying screen depths, ranging from 4 planes (4 bits per pixel) to 32 planes (32 bits per pixel). Typically, higher-end graphics devices have higher-depth graphics outputs, especially for 3-D graphics, and the lower-

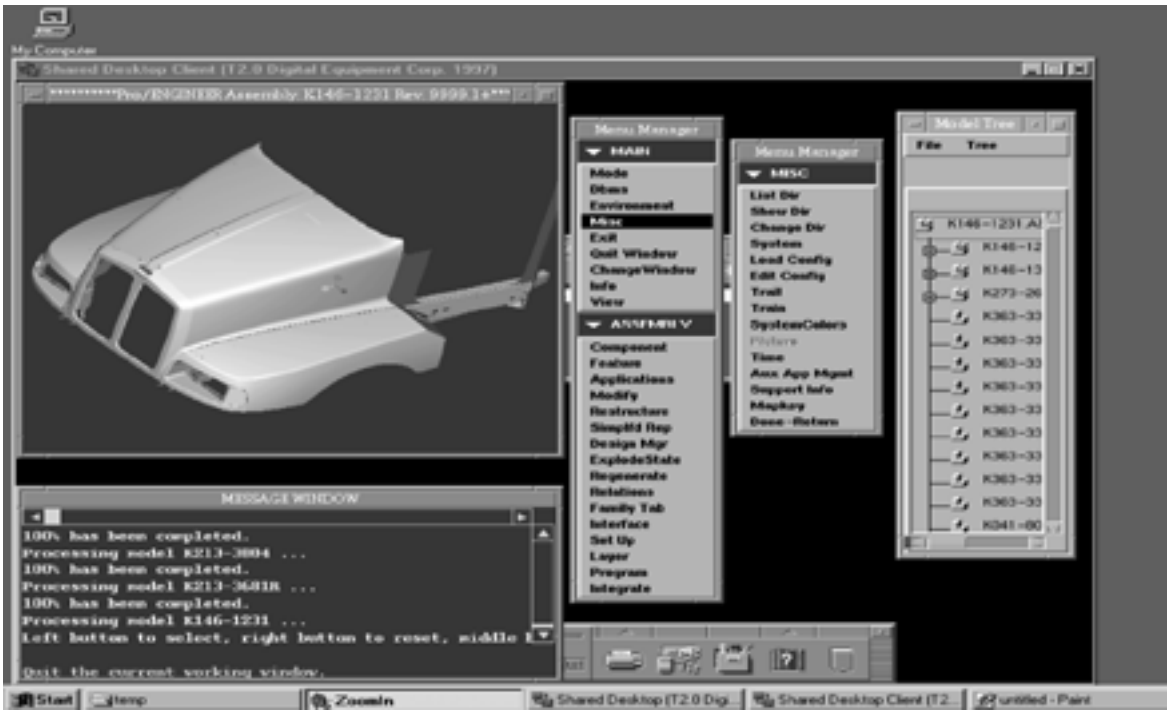

Figure 4
Screen Capture of the Windows Shared Desktop Client Sharing Pro/ENGINEER with a UNIX Shared Desktop Server

depth displays are usually found on less-capable, 2-D graphics platforms. Most laptop computers have low bit depth (8 to 16) displays and no 3-D capabilities. Commodity PCs also typically have 8- or 16-plane depths. Graphics devices that support 3-D graphics provide deeper display types such as 24-bit or 32-bit. Some devices support a mix of several or all the bit depths listed in the matrix (below) either concurrently or for the entire screen at one time.

We defined a matrix of screen depths and proceeded to fill in the various combinations so that the application would work effectively across different platforms and graphics hardware capabilities. The matrix enables computers without 3-D capability to display the output from 3-D-capable graphics devices. The matrix of screen-depth combinations follows.

Output Bitmap Depth	Input Screen or Visual Type Depth						
	4	8	12	15	16	24	32
4	x	md	md	d	d	d	d
8	e	mx	md	d	d	d	d
12	n	n	n	n	n	n	n
15	e	me	me	x	d	d	d
16	e	me	me	e	x	d	d
24	e	me	me	e	e	x	d
32	e	me	me	e	e	e	x

The matrix shows input screen or visual type depth across the top row and delineates output bitmap depth on the left column. Bitmap depths of 4, 8, 15, 16, 24, and 32 are used in Windows systems, and depths of 4, 8, 12, 24, and 32 are used in X11. The *x* in the matrix requires no conversion and is captured and displayed without the need for additional conversion. The *e* shows bitmap depths that can be expanded to the output format by using a colormap or by shifting pixels into the correct format. The *d* shows that information must be dithered to match the output. Dithering can result in a minimal loss of information, but we have developed a very good and efficient method of doing this conversion. The *m* (mix mode) marks those visual types on X11 that can exist on the screen when the root depth is 24 or 32; i.e., an 8-bit window can be present on a 24-bit display. The mix mode requires a different interpretation of the 24-bit pixels prior to compression and transmission. Since no 12-bit output displays exist, *n* marks inapplicable transformations. Alternate formats of 24 pixels (3 bytes per pixel and blue/green/red [BGR] triples) are supported as well as 8-bit pseudocolor and 8-bit true color.

Sample Uses

Like other collaboration software, the Shared Desktop application can be used in remote situations to help

people communicate and share data. These uses include telecommuting, debugging/support, and education.

Telecommuting

One feature we built into Shared Desktop is the ability to originate a sharing session from a remote location. Our intent was to allow an individual to work outside the office environment on a home PC or a laptop computer. In the telecommuting scenario, a workstation with high-end graphics functions and applications located in the office would call back the home user's low-end system and present the user with his work environment. For example, consider a user of PTC's Pro/ENGINEER who is working on a 3-D assembly with a 100-MB database and must make a change to the part from home. Prior to the Shared Desktop application, the only options were either to mimic the work environment at home or drive to the office to make the change. To mimic a work environment, the equipment at home must support Pro/ENGINEER software and might require 3-D hardware. In addition, the user would have to retrieve a recent version of the 100-MB database over the telephone lines, which would take many hours to copy. With the Shared Desktop application, the user can access the 100-MB part using the low-end computer over standard telephone lines. The changes to the assembly then occur on the system and to the large database at the office.

Remote Debugging/Support

Another use of the Shared Desktop application is for customer support or remote debugging. Consider the user of a 3-D design application who discovers a bug in a new version of the software. A complex model often causes a bug that requires software support to obtain the database to re-create the problem. Using Shared Desktop, a user could show a support representative the problem on the running application, as opposed to filing a problem report.

Off-site Training

A remote training scenario provides a final example of collaboration using computers. The Shared Desktop application facilitates remote training by connecting students in a sharing session. Each student's desktop displays a lesson composed of the course material installed on the instructor's desktop. Students interact with the teacher by audio, mouse, and keyboard actions on objects in the screen viewport. In essence, the teacher uses the synchronized cursors to highlight or point to objects on the screen.

Conclusion and Future Directions

The Shared Desktop collaboration software employs a simple user interface that emphasizes ease of 3-D application sharing and audio conferencing. Compared to application sharing based on a protocol splitter, the Shared Desktop application offers easier interoperability and better latency during 3-D operations. With a protocol splitter approach, it is difficult to decide which, if any, graphics events to drop when network jitter or network bandwidth delays occur. Our approach is synchronized to the last screen capture. When the network is no longer congested, the current screen capture can be sent, thus minimizing the perceived effect of the network delay. The only disadvantage to bitmap sharing is its requirement that the windowing system and display driver implement a DMA screen capture function and not programmed I/O. DMA screen capture requests have a minimal load on the windowing system.

We are planning a number of improvements to the advanced development version of Shared Desktop. In our initial work, we made no changes to the windowing systems. Ideally, the product version might have a mechanism that notifies an application when and where another application has made changes to the screen. With the added ability to capture only those areas of the screen that have changed since the last notification, the windowing system could perform the first two steps in the capture process.

Although the compression scheme we implemented works for most cases, some graphics may not compress well using the combination of RLE and LZ77. Instead, content-specific compression or adaptive compression techniques might be better applied. This is an area of study we hope to pursue.

The current graphical user interface (GUI) lacks some conferencing features. The product version will be packaged with other applications to provide video, chat, whiteboard, file transfer, and user locator/directory services.

Finally, the sharing model we implemented for the Shared Desktop application is easily ported to other systems. Thus the application could be available for widespread use.

References

1. A. Hopper, "Pandora—An Experimental System for Multimedia Applications," *Operating Systems Review,* vol. 24, no. 2 (1990): 19–35.
2. J. Neider, T. Davis, and M. Woo, *OpenGL Programming Guide* (Reading, Mass.: Addison-Wesley Publishing Company, 1993).
3. *Visual C++ Version 4.0 SDK Programmer's Guide RISC Edition* (Redmond, Wash.: Microsoft Corporation, 1995).
4. *Multipoint Still Image and Annotation Protocol,* ITU-T Recommendation T.126 (Draft) (Geneva: International Telegraph and Telephone Consultative Committee, 1997).

5. R. Scheifler, J. Gettys, R. Newman, A. Mento, and A. Wojtas, *X Window System: C Library and Protocol Reference* (Burlington, Mass.: Digital Press, 1990).

6. *Pulse Code Modulation (PCM) of Voice Frequencies,* CCITT Recommendation G.711 (Geneva: International Telecommunications Union, 1972).

7. R. Palmer and L. Palmer, "Video Teleconferencing for Networked Workstations," United States Patent no. 5,375,068 (1994).

8. P. Heckbert, "Color Image Quantization for Frame Buffer Display," *Computer Graphics,* vol. 16, no. 3 (1982).

Biographies

Lawrence G. Palmer
Larry Palmer is a consulting engineer in Workstations Graphics Development. He joined DIGITAL in 1984 and currently co-leads the Shared Desktop project. He holds a B.S. in chemistry (summa cum laude) from the University of Oklahoma and belongs to Phi Beta Kappa. He is co-inventor for nine patents on enabling software technology for audio-video teleconferencing.

Ricky S. Palmer
Ricky Palmer is a consulting engineer in Workstations Graphics Development. He joined DIGITAL in 1984 and currently co-leads the Shared Desktop project. He holds a B.S. in physics (magna cum laude), a B.S. in mathematics, and an M.S. in physics from the University of Oklahoma. He is co-inventor for nine patents on enabling software technology for audio-video teleconferencing.