Implementing Gesturing with Cursors in Group Support Systems

STEPHEN HAYNE, MARK PENDERGAST, AND SAUL GREENBERG

STEPHEN HAYNE is an Associate Professor in Management Information Systems at the University of Calgary. His research interests lie mainly in distributed database design, software engineering, knowledge-based technology and human–computer interaction in group support systems. To this end he has implemented tools in graphical environments to assist groups in communication and decision making, i.e., shared drawing, group brainstorming, concurrent issue surfacing, and consolidation. His research has been published in *Information and Management* and the *Journal of Information and Technology Management*, as well as in several major conferences. He is an editor of *Groupware for Drawing and Writing* (McGraw-Hill, forthcoming).

MARK PENDERGAST is a Research Director in the Center for Information Management, Management Information Systems Department, at the University of Arizona. His research interests include computer-supported cooperative work, data communications, software engineering, process re-engineering, and group support systems in human–computer interaction. He has worked as an analyst and engineer for Control Data Corporation, Harris Controls, Ventana Corporation, and as an Assistant Professor at the University of Florida. His work has appeared in several books, journals, and was presented at numerous conferences.

SAUL GREENBERG is an Assistant Professor of Computer Science at the University of Calgary. Dr. Greenberg is a researcher in the areas of computer supported cooperative work (CSCW) and human–computer interaction (HCI), with special interests in groupware design and architecture, customizable interfaces, and information visualization and management. He is the author and an editor of several books, including *The Computer User as Toolsmith* (Cambridge University Press, 1993), *Computer Supported Cooperative Work and Groupware* (Academic Press, 1992), and *Groupware for Drawing and Writing* (McGraw-Hill, forthcoming). He serves on numerous program committees in the CSCW/HCI area, and is on the editorial board of the *International Journal of Man–Machine Studies.*

ABSTRACT: Gesturing from one human to another appears to span all cultural boundaries; one could possibly call it a universal means of communication. Group work studies have shown that gesturing makes up over 35 percent of all interactions. Participants use hand gestures to enact ideas, to focus the attention of the group, to signal turn-taking, and to reference objects on the work surface. Specifically, this paper explores gesturing as applied to users of group support systems. We address practical issues such as: at what level of interaction might gesturing be supported, how large and what shape should gesture pointers be, how they should move, network and processor throughput requirements, and group size effects. Our results show that while

Copyright © M.E. Sharpe, Inc., 1994

full motion computer gesturing can be supported on PC-LAN systems for small groups, gesturing for medium and large groups requires the use of special techniques such as regulating transmission rates, motion smoothing, and point and quiver cursors. These techniques could also be applied to wide area network implementations to reduce network traffic and latency problems.

KEY WORDS AND PHRASES: computer-supported cooperative work, gesturing, groupware, local area networks, telepointers.

1. Introduction

GROUP SUPPORT SYSTEMS IS AN EMERGING AREA of research that spans group decision support systems (GDSS) and computer-supported cooperative work (CSCW). GDSS is "an interactive computer-based system that facilitates the solution of unstructured problems by a set of decision-makers working together as a group"[4]. The design goal is to increase the efficiency and effectiveness of meetings. The usual method is to "manage" the interactions between group members by enforcing rules of meeting protocol and structure [18]. Existing GDSS tools facilitate both small (3–6 members) and large (7–30) groups through the various stages of the decision-making process [3, 33].

On the other hand, CSCW is defined as "the study and theory of how people work together, and how the computer and related technologies affect group behavior" [24]. CSCW implementors build "computer-based systems that support two or more users engaged in a common task (or goal) and that provide an interface to a shared environment" [7]. These systems usually facilitate communication between members of a small group and provide task-specific tools. The software rarely regulates the actual meeting process; the designers expect that normal social protocols between participants will suffice.

It is our experience that meetings in most corporate situations lie somewhere between the domains of CSCW and GDSS; not all meetings result in decisions nor do all group processes center around a task. Sometimes structured processes are crucial at meetings; at others they are debilitating. Rather than split semantic hairs about where our research resides, we will use the label group support systems (GSS). This research entails understanding how groups interact; applying that understanding to designing systems that help people work together; and observing people using our systems. The last point often constrains our implementations to standard delivery platforms (such as IBM PCs), and much of our effort is devoted to overcoming the platform limitations. The focus of this article is on gesturing: why it is required, how it can be implemented on a computer, and what pragmatic decisions must be made during implementation.

2. Gesturing Defined

> *gesture*: 1) the use of motions of the body as a means of expression, 2) a movement of the body that expresses or emphasizes an idea, a sentiment or attitude [23].

SOME MANNER OF GESTURING TAKES PLACE THROUGHOUT ANY INTERACTION between human beings. Gesturing appears to span all cultural boundaries; one could possibly call it a universal means of communication. Gesturing not only occurs between people, but also between people and their artifacts. In an ethnographic study of eight short-term, small-group design sessions, Tang built a descriptive framework that categorized activities over a shared work surface (large sheets of paper tacked on a table or white board)[28, 29]. These primitive actions were observed, and were combined to mediate several essential functions:

Action:

- *listing*: produces nonspatially located text or alphanumeric notes;
- *drawing*: produces graphical objects, typically a two-dimensional sketch with spatially located textual annotations;
- *gesturing*: purposeful body movements that communicate specific information such as pointing to an existing drawing.

Function:

- *storing information*: refers to preserving group information in some form for later recall;
- *expressing ideas*: involves interactively creating representations of ideas in some tangible form;
- *mediating interaction*: facilitating the collaboration of the group including turntaking and focusing attention.

Hand gestures played a prominent role in all work surface activities (about 35 percent of all actions). Participants gestured to enact ideas, to signal turn-taking, to focus the attention of the group, and to reference objects on the work surface [28, 29]. Clearly, gesturing is of paramount importance in group interaction. Tang demonstrated this for small design groups, and it is our belief that gesturing is essential for almost all group activities involving a shared work surface.

We are exploring hand gesturing over displays in group support systems. Obviously, as soon as group work is moved from a "manual" face-to-face environment to a dispersed "computer-supported" environment, many questions arise about gesturing. How can hand gestures be tracked as input? How do we display a gesture on a screen? If there is to be a gesture icon or cursor, how large must it be? How should it move? What happens as group size increases (scalability)? What network bandwidth is needed? We will address these questions in section 4, after first reviewing existing implementations of gesturing in section 3. Our experiences implementing different gesturing strategies via multiple pointers on an LAN-supported IBM PC platform are described in section 5, which is heavily illustrated with performance benchmarks.

3. Prior Research on Gesturing

WE ARE CONCERNED WITH ENHANCING A GROUP'S real-time interaction when using groupware tools in a computerized shared visual work surface, where one's actions are immediately visible to all [7]. The displays are software equivalents to whiteboards and flip charts; the tools could be text editors, hypertext, structured drawing tools, and so on.

Many collaborative systems do not support gesturing. For example, we now see teleconferencing setups that combine voice conferencing with facsimile transmission, electronic white boards, and even slow scan video. While all participants can see the same resulting information, gestures referring to an artifact are only visible through indirection or not at all. For example, if one points to a drawing on a document at one site, a person at the other site must find his or her equivalent document and determine where the reference is. Similarly, one rarely sees the fine-grained process of others creating and manipulating artifacts. While people can get their job done without gesturing, we argue that much of the conversation is used as a substitute to gesturing, for example, "I'm pointing to the second line in the third paragraph," and that breakdowns are frequent, for example, "Which one do you mean?"

One of the first systems to support gesturing explicitly was *BoardNoter*, a computerized whiteboard used to support face to face meetings [27]. Gesturing was through a single large "telepointer." One person at a time could grab and control this special cursor, which was then seen on every display. All other individual cursors remained invisible to the group. In addition to tracking the cursor, clicking the mouse button would cause a static image of the pointer to be deposited (and remain) on the board [26]. Several commercial *view-sharing systems*, which allow people to share standard single-user application through serial interaction [9], can also support single cursors. Unlike *BoardNoter*, there is no difference between the local cursor and telepointer, for example, in Farallon's Timbuktu [8]. The common cursor responds to each user's mouse movement. When several people move their mice, the result is "cursor wars."

The first serious works in gesturing, all motivated by Tang's work (described above), were implemented using two different technologies: video fusion and computational space. *VideoDraw* was the original video-based system [30]. Each person has a monitor that displays the image of a camera pointing to the other person's monitor. Participants draw directly onto the monitor, and the camera captures both the other person's hand and the drawing underneath. Feedback is eliminated through polarizing filters. The result is that the shared workspace contains (from one person's point of view) his or her physical hand, pen marks on top of the screen, plus the other person's hand and marks in the image. *TeamWorkStation*, on the other hand, uses hardware to fuse video signals [16]. The advantage here is that people can perform their activity on any work surface (such as a desktop), with a video camera recording and fusing its image with the work surface image of the other participant. Finally, *VideoWhiteboard* allows each user to see the drawings and a shadow of the gestures of collaborators at the remote site [31]. Here, people see their partners and movements as a silhouette appearing on the other side of a translucent whiteboard. These video systems are limited. First, participants cannot manipulate other's marks, since they only see them as a video image. Second, these systems are not scalable since serious image deterioration results when too many images are fused.

At the same time, several computational systems were developed that implemented gestures through multiple cursors. *GroupSketch* is a computer-based group sketchpad where one or more people can simultaneously draw, type, and gesture around the display [10, 11]. Its cursors were designed specifically for gesturing around the following criteria:

1. Cursors must have enough prominence on a multicursor display to attract the attention of other participants. A large 64×64 bit cursor was used instead of the traditional 16×16 bit cursor.
2. Since gestures must be seen in order to convey information, all cursors within a work surface are always visible to all participants.
3. Cursors are unique, each identifying the person to whom it belongs. While face-to-face gesturing has natural cues to help identify who is gesturing, cursors do not. *GroupSketch* labels each cursor with its owner's name and each new cursor is rotated 90 degrees from the last (while the idea of rotation was first put in to reflect 'fferent seating orientations of participants around a drawing, its real value was that it allowed up to four cursors to touch the same pixel without overlapping each other).
4. Cursors always maintain their same relative location on every display so that they retain their relation to the work surface objects.
5. Cursor movements appear continuously and with no apparent delay on all displays, which means that they remain synchronized with verbal communication.
6. Cursors change their shape to reflect a natural action. Four gesture modes are supported (pointing, writing or drawing, erasing, and directing attention) by distinct cursor shapes (a pointing hand, pen, eraser, and large arrow, respectively).

In parallel with *GroupSketch* came the very similar *Commune* [2, 15, 17]. Instead of a mouse and a vertical display, *Commune* users were able to write directly on a flat screen with a pen. This provided them with improved control or display compatibility and fine motor coordination. Many second-generation sketching packages now support gesturing, such as *XGroupSketch* [11], a functionally richer window-based version of *GroupSketch*; *WScrawl* by Brian Wilson [34], *GroupScratchPad* (GSP) by Stephen Hayne [12], and *ShDr* by Paul Dourish [5]. We are also seeing these ideas transferred to the domain of structured drawing: *GroupDraw* [11], *MMM* [1], and *MUGE* [20].

Several researchers are now investigating "gaze awareness," where a person can see where on the work surface his or her partner is looking [14]. Although we do not pursue it further here, a person's gaze is a gesture as well. By tracking other people's eye movement over the work surface, we see their foci of attention and their degree of interest, and we obtain cues for topic switching, and so on. The few computational attempts at supporting gaze awareness rely on overlaying the work surface on top of a video image of a person's face, and arranging cameras and half-silvered mirrors in such a way that eye references to the work surface are consistent across sites. Again, this video configuration may not be scalable to larger groups.

Some computer tools for tasks other than drawing also support gesturing. In *ShrEdit*, a shared text editor, all insertion "carets" of participants are visible, as well as the text that they type [19]. In *rIBIS*, a hypertext system supporting argumentation, there are two modes of operation: loosely and tightly coupled. In the latter, a single group cursor available through turn-taking is visible to all, thus strengthening the feeling of participation [22]. Both the *Multi-User Graphical Editor* (*MUGE*) and the *Graphical Issue Analyzer* (*GIA*) by Pendergast and Hayne [13, 20] have implemented specific gesturing cursors (discussed in the next section).

In summary, existing systems support gesturing through a handful of strategies. These include a single static pointer that is placed on the screen but does not track mouse movement, a single dynamic pointer that follows the mouse, multiple pointers with fine-grained tracking, and video images of actual hands as well as exploring gaze awareness. Of course, there are also systems that do not support gestures at all, but we believe these provide an impoverished work surface for group members [11].

4 Design Considerations for Gesturing

HE AIM OF OUR RESEARCH IS TO FOCUS on a computer-based mechanism for supporting telepresence in the form of gesturing. Telepresence attempts to give meeting participants the feeling that they are all in the same room [6] by sharing explicit and implicit participant dynamics. Paramount among these actions are gestures and metalevel communications. As we are not targeting a multimedia environment, we have chosen to study the requirements for gesticulation as implemented by multiple cursors that can be displayed on every participant station in real time.

This section describes some of the design considerations we have encountered in our work on gesturing. It is presented as a loose framework revolving around four aspects: different ways cursors can express gesturing through motion; the appearance of a gesture cursor; effects of group size and distribution on cursors; and consequences of relaxed "What you see is what I see" (WYSIWIS) [26]. We assume a voice channel is available to the group, and that multiple cursors are preferred over single cursors.

4.1. Gesturing through Cursor Motion

The presence of a cursor on the screen indicates who is active in the session. The motion of the cursor indicates a pure gesture or a gesture-based activity. We believe that the best gesturing systems will support real-time motion on all screens. Unfortunately, some platforms may not allow full motion because of limited network bandwidth, latency, and slow processor speed. The following strategies for supporting low to high bandwidth cursor motions are presented in order of lowest to highest data and processing requirements. The implementation model used is that each workstation sends a message on its cursor status to the other participants' workstations.

Point: When the user points and presses a mouse button, a single message is sent to all the remote users. The gesturing cursor leaps to the new location from its old one and remains there until its owner sends a new location message. The problem is that a user may not see the sudden transition, thus not noticing the gesture.

Point and quiver: In order to attract the viewer's attention, the above strategy is modified so that the receiving station *jitters* the cursor back and forth in its new position for a period of time.

Limited motion: Only a certain number of gesture messages per second are sent between stations. As the number is increased, the cursor motion moves from being stilted and jerky to a smoother animation. For example, *ShDr* [5] allows the user to control the number of transmitted cursor movements through a slider in the window's control panel. The fewer messages sent, the less accurately the motion reflects the gesture. Also, the jerky cursor is visually unattractive.

Limited motion with smoothing: After receiving a new cursor position from the sender, the receiving station computes several points between the previous and the current location, and then animates the cursor through these intermediate points. While this eliminates jerkiness, it still does not guarantee that the path reflects the actual movement of the sender. Note that smoothing can also be applied to point, and to point and quiver. This technique is applicable to slow networks, but does not overcome slow processors.

Full motion: Every time the cursor is locally displayed in a new location, a gesture message is sent. Every movement of the cursor, no matter how small, is immediately broadcast. If the network and processor are fast enough, every receiver will display the cursor exactly as the sender does. For many years the motion picture industry used 16 frames per second as the standard for smooth animation. We feel that 10–20 messages per second will support full motion. While sending more messages may be possible, it is probably not necessary.

Which is the best choice? We believe that full motion gesturing is preferred for most situations. However, there may be special occasions when a user would want to leave a pointer parked at a specific location, such as when long network latency (> 0.5 seconds) is combined with real-time voice. Full motion may be unacceptable when gestures do not match the spoken communication.

4.2. Cursor Appearance

Single cursors are usually presented on a screen as a small (16×16 pixel) bitmap, usually a left-leaning arrow. A naive approach to multiple cursors is simply to copy the appearance of single cursors. However, several issues appear that demand a slightly more sophisticated approach.

Cursor size: Cursors in most single-user graphical user interfaces are quite small. As screens get larger and the display busier, the cursor becomes difficult to find. Most people just jiggle their mouse (we can easily spot motion). Some window systems have tricks for focusing a user on the cursor location. Examples of the latter are "Xeyes" (two graphical eyes track the cursor), and "oneko" (where a kitten chases the cursor around the screen, eventually to sleep on top of it when idle). When multiple people are using a display, the cursor size becomes much more critical. Since we maintain that cursors are critical to gesturing, a larger size (say 64×64 pixels) would be more appropriate for small-group interactions [11, 20]. The trade-off is that cursors can quickly consume too much screen real estate as group size increases (described later). While we do not yet know what the optimum size is, we caution implementors against choosing the standard 16×16 pixel size just because they are accustomed to it.

Owner identification: Assuming nonanonymous interaction, remote cursors must be easily discernible from the local cursor, and participants should easily be able to distinguish who is doing what. In the first case, color or slight shape changes can be useful. For example, the local cursor might be black (or solid) while the remote cursors are colored (or outlined). In the second, the cursors can be uniquely labeled by a participant's name so that participants can quickly determine who is doing the gesture. In practice, this may be easy, since user actions are often tied to voice [11]. Although labeling may be left off if some degree of anonymity is desired, participants will likely be able to identify its owner by other clues.

Cursor lifetime: While it can be useful to have cursors displayed at all times (especially to indicate who is present), inactive cursors can become passively intrusive in larger groups. To address this, remote cursor display can either be put under individual control or can have a decay function over activity and time. Two interesting approaches to decay are to have a gravity function which removes the inactive cursor to a corner of the screen called a parking area, or to have the cursor merely dissolve away [12].

Cursor shape: Since many different and natural actions are to be expressed, the cursor shape must change depending on the action—this will add to the quality of the gesture. Both *pointing* (e.g., finger) and *attracting attention* (e.g., flashing arrow) have been identified as crucial general actions [10, 11, 20]. If the application requires drawing or listing, the cursor should also reflect this by perhaps using a pencil, a caret, or other shapes that indicate the action. Gratuitous cursor shapes, as implemented in the *Aspects* groupware system [32] likely have no positive effect.

4.3. Group Size and Distribution

Clearly, cursors convey metalevel communication information, that is, how many people are in the meeting and how their activity is balanced. As the group size increases, the number of cursors appearing on the screen could begin to intrude on the interaction. One approach to managing this problem is to reduce the size of the remote cursors using a step function. As examples:

1. For one to four people, use the "regular" size of 64×64 pixels;
2. For five to six people, reduce the cursor size to 32×32;
3. For seven to eight people, reduce the size to 16×16.

For larger groups, each user may be displayed as a single pixel on the screen—this will still provide participants with "gestalt" of the activity level. This step function was implemented in the latest versions of *GroupSketch*, *MUGE*, and *GroupScratch-Pad*. Cursor size is altered by the software as people enter and leave the session. This notwithstanding, we feel that when groups become very large, the lifetime of cursors will need to be adjusted.

Decreasing the size of the cursor severely limits the amount of information it can display. Very small cursors do not have room for a label, so identification becomes problematic. Also, as cursor numbers increase, it will become difficult to tell who is doing something important, and whose voice is tied to which cursor. We believe that the cursor size should be altered to reflect important actions. For example, the cursor size in a medium-sized group of eight people could be quite small (4×4 pixels) [11], but users might press a mouse button and get a large labeled pointing arrow especially designed to attract the group's attention. Similarly, whenever someone starts to draw or type, their cursor grows. Gesturing could be used to get access to the "floor" or voice channel when individuals are dispersed geographically [25], particularly useful for sessions using conference calls. Perhaps a "hand-raising" cursor is required.

4.4 Relaxed WYSIWIS

Much of our discussion assumes strict WYSIWIS, where everyone sees exactly the same view. Since we feel that most group software will eventually relax WYSIWIS along congruence of view, transformation of cursor coordinates between the sending and receiving views must be performed. As well, screens have differing resolutions and if a windowing environment is used, windows can be placed on different parts of the screen. This transformation is easily done by mapping the cursor location to a world coordinate system at the sender and (1) if that location is outside the receiver's view, discard the message; or (2) map the world coordinate to the receiver's viewport coordinates. It is extremely important that when one user points at a place in his or her window or screen locally, the same logical place is "gestured" at on the remote stations.

Of course, this leads to a variety of human factors issues. Consider figure 1, which shows four users actively working on a shared canvas. From user A's perspective, B's view completely overlaps, C partially overlaps, and D is disjoint. First, should A have peripheral awareness of the others in the virtual canvas, particularly those who are completely outside A's viewport (D)? In *Shared ARK* and *MUGE*, for example, the display includes a miniature of the entire canvas that shows the boundaries of each person's area of view [20, 25]. This conveys some of the metalevel information described earlier. Second, if viewports overlap (A and B), what happens to the gesturing cursor when it moves across a view boundary? For example, if B moves the cursor to the lower right of the screen, would it suddenly disappear from A's view (as in *XGroupSketch*), or would it "linger" in the last visible spot (as in *GroupScratch-Pad*)? What consequence does this have on gesture interaction? Third, how should views be linked together? Assuming that users will move from a loosely coupled mode where they are working more or less independently to a tightly coupled mode throughout a session (and back), there should be ways to link views and gesture spaces together. For example, *MUGE* and *GSP* include the notion of leaders and followers, where users have the option of making their viewports and scrolling actions slaved with one another [20]. A user can be both a leader and a follower; leaders take followers along with them as they move about the work surface (zooming or scrolling).

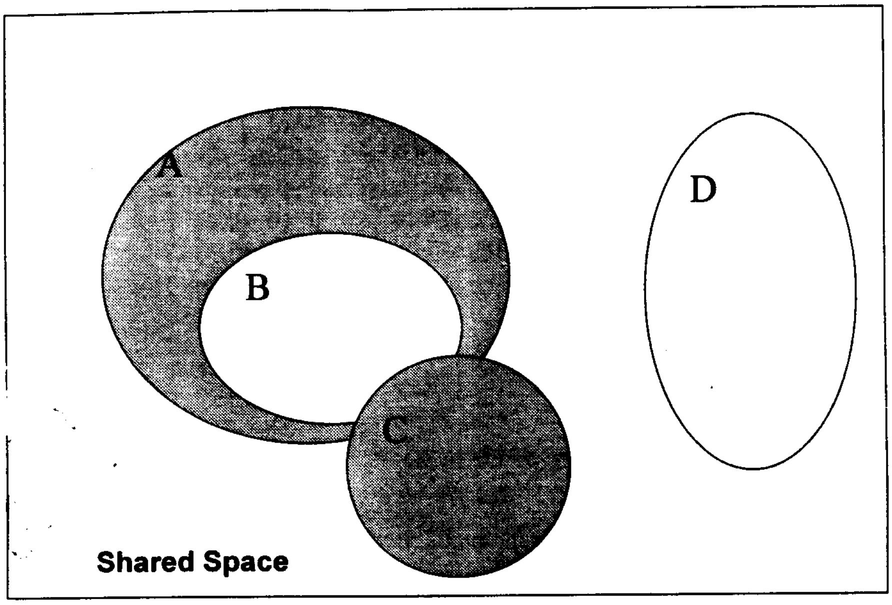

Figure 1. Translation between Viewports A, B, C, D

5. *Performance Testing*

SUCCESSFUL IMPLEMENTATION OF FULL MOTION GESTURING IN GSS applications is dependent on the performance characteristics of the network and host operating system. Many of the previous systems have been implemented on UNIX hosts, but we feel that the more realistic corporate environment of IBM PCs and a graphical user interface would represent a lowest common denominator. Thus, we conducted our tests in the University of Calgary Norcen Meeting room which has 34 IBM PS/2 Model 55 (386sx/16) PCs networked using both a 16MB Token Ring LAN and a 10MB Ethernet LAN supported by a Novell Netware (V3.21) file server.

Microsoft Windows 3.1[1] applications were used and the communication was managed by the Network Interface Object (NIO) as described by Pendergast [21]. The NIO manages application layer communications (OSI) by providing a "channel" object that facilitates the creation of GSS applications by providing virtual, multicast connections that parallel the meeting room metaphor of communications. Multicast connections permit communication between two or more stations using a single network name or address, thus not requiring multiple separate session connections with each one. The NIO implements multicast channels using two transport level protocols: (1) reliable session connections, and (2) broadcast datagrams. Depending on the nature of the data, the applications can dynamically select either the reliable (session connection) or unreliable (datagram) path when sending the message. Using the reliable channel requires the NIO to transmit the message to each station individually. Broadcast datagrams are sent just once, independent of the number of stations participating. The unreliable path has limited message size (approximately and is not acknowledged, while the reliable path has no size limits and are guaranteed sent until a time-out occurs.

The NIO is incorporated into several GSS applications, including *MUGE, GIA, GSP*, and *Graphical Brainstorming* [12]. Since all Windows (or DOS) based applications will need to use a network layer (perhaps like the NIO), we felt it necessary to determine if typical LANs and PCs could support gesturing.

5.1. Transmission Time

This test determined the time required to transmit messages of various sizes over the unreliable (broadcast datagram) and reliable (multicast) communication channels. The test consisted of a station sending 100 messages as fast as possible and recording the elapsed time. The number of receiving stations was varied from 1 to 12 for message lengths of 100, 1,000, and 2,000 bytes, thus requiring 1, 2, and 4 network packets respectively. Table 1 presents the elapsed time, in seconds, to transmit 100 messages. Figure 2 illustrates the performance differences between the multicast and broadcast datagram channels.

The transmission times for the datagram channel remained constant while the multicast channel's rate increased linearly with the number of receiving stations. This is as expected since the multicast channel transmits a message to each receiver separately. The amount of time required to transmit (multicast) a 100-byte message to 12 stations is 0.25 seconds and we predict that for 24 stations it would be 0.50 seconds. This makes a clear case for the use of unreliable datagram communications for gesture messages even though some datagrams may be lost.

5.2. Datagram Throughput

This test was conducted in order to determine the maximum rate that a station could transmit datagram messages of a size appropriate for gestures. A single station (IBM PS/2 55) transmitted 100 datagram messages of 100 bytes as fast as possible and recorded the elapsed time. Using the Novell Netware IPXSPX Windows interface, this time was measured at 2.314 seconds (average), or 23 milliseconds per message (43 messages/second). To measure the overhead of the test program and high-level communications handler (NIO), the actual datagram transmission was then disabled and the test repeated. The resulting overhead was approximately 5 milliseconds per message, leaving 18 milliseconds for the actual transmission. This seemed to be slow considering a 16MHz processor and a 16MB Token Ring LAN. This same test was also performed using the 10MB Ethernet protocol. The results from this test were better. It took 14 milliseconds per message, with 5 milliseconds of overhead. However, 9 milliseconds (Ethernet) and 18 milliseconds (Token Ring) still seemed excessive.

To discover if the Windows operating environment had an effect on transmission rates, a DOS test program was created that measured the time required to transmit 100 datagrams (100 bytes each). This program was able to transmit the 100 messages in less than 300 milliseconds (Token Ring) and 100 milliseconds (Ethernet), or about 3

Table 1 Transmission Times in Seconds for 100 Messages

Receivers	Multicast 100 bytes	Multicast 1,000 bytes	Multicast 2,000 bytes	Datagram 100 bytes
1	2.257	4.399	8.739	1.982
2	4.729	8.628	17.032	1.928
3	6.871	13.407	25.380	2.477
4	8.463	17.142	33.564	2.422
5	10.608	21.152	42.408	2.532
6	12.638	25.436	50.427	2.587
7	14.890	29.665	58.611	2.615
8	17.032	34.004	66.904	2.642
9	19.037	38.041	75.363	2.395
10	21.041	42.078	83.821	2.148
11	23.101	46.362	92.307	2.065
12	25.161	50.646	100.792	1.982

and 1 milliseconds per message respectively (one-sixth the time of Windows). Due to the extreme difference between Windows and native DOS, we wanted to pinpoint the problem in Windows by eliminating the Novell interface to the network and substituting the IBM NETBIOS interface. This third test gave results identical to the IPXSPX version. We believe we have verified the existence of a Microsoft Windows-induced delay in the transmission process. One source of the delay comes from having to transfer data packets and command blocks from extended memory to low DOS memory where the adapter can access it. We also suspect that the Windows kernel only processes interrupts after a certain arbitrary number of clock ticks. Reducing the NIO's overhead by 75 percent would allow transmission of over 50 datagrams/second—far more than we feel is required for gesturing. But discovering the reason for the delay inside of Window's kernel and reducing it by just 20 percent would yield the same results.

5.3. Multiple Sender Interaction

A third set of tests determined the network interaction effects that would occur if several stations were gesturing at the same time. In this test the number of sending stations was varied from 1 to 4 and the message transmission rate was varied from 5 to 25 messages per second. Each message was 100 bytes long and sent using broadcast datagrams. The results of the test revealed two important thresholds: one when the network protocol began favoring one station over another and a second when datagrams began to be lost. These thresholds are shown in figures 3 and 4.

Datagram packets are lost whenever there is a transmission error or if a station does not have an outstanding receive request. Since real-time gesture messages will probably be transmitted at a rate greater than 10 per second, we believe the loss of a few is not a major concern. The favored station threshold does present a problem. Table 2 presents the data from a 20-second run where four stations were transmitting (and receiving) at the same time. Station A was not able to transmit at the same rate

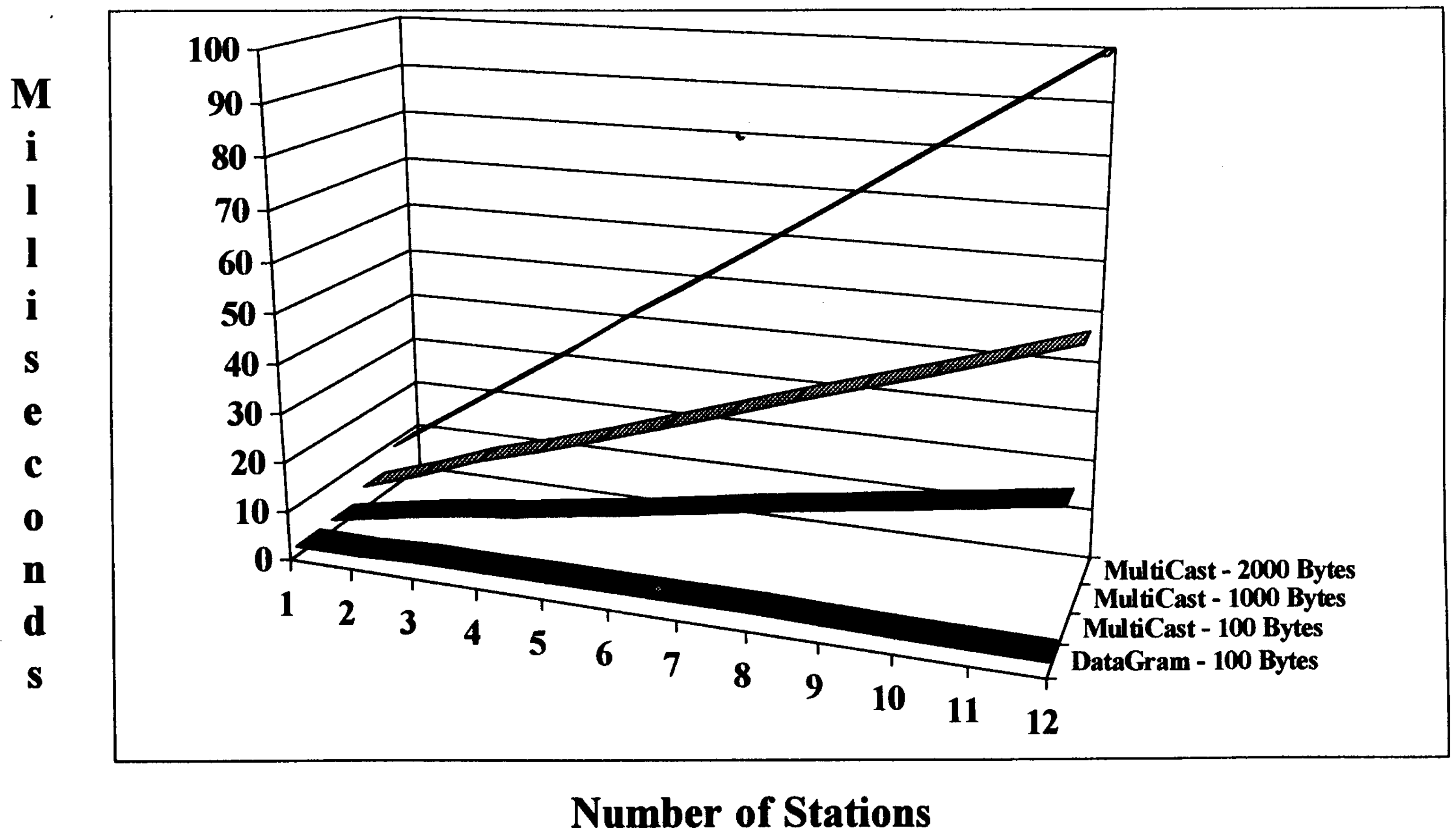

Figure 2. Performance Differences between Multicast and Broadcast

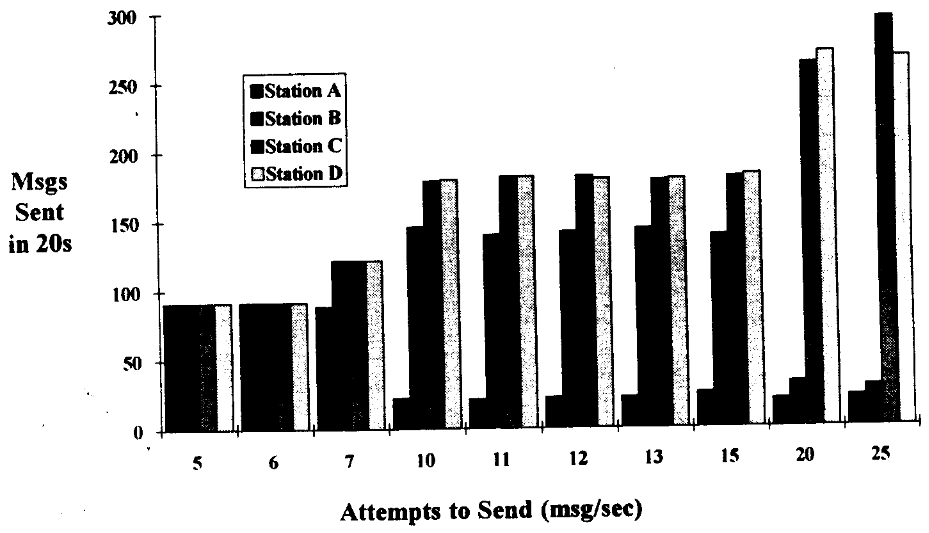

Figure 3. Network Favoritism Threshold

as the other stations once the send rate was increased to 7 messages per second. Station B began being penalized at 9 or 10 messages a second, while stations C and D were able to transmit at a high rate for all speeds. At the speeds required for smooth gesturing (minimum of 10–15 per second), station A was only able to transmit one message per second. This results in a very jerky gesture motion. Since a station's ability to transmit/receive is based on processor speed and location on the ring, certain stations will always be at a disadvantage. This indicates that the software controlling the gesturing should be intelligent enough to govern its transmission rate whenever two or more stations are gesturing at the same time. This test was repeated on the Ethernet LAN, and yielded similar results.

5.4. Video Display

A final performance test was conducted in order to measure the processor overhead required for displaying different types of gesture pointers. This test consisted of drawing and erasing a 24×36 pixel bitmap or a five-sided filled arrow polygon, of roughly the same area, 2,000 times. We felt that the overhead of a bitBlt call might be larger than the overhead of a drawPolygon call. The test was run on four different CPUs (see figure 5). For all CPUs, the drawing of the polygon was four to five times faster than the bitmap. The slowest CPU (Model 55) required 14 milliseconds for the bitmap and 3.8 milliseconds for the polygon, the fastest CPU (Gateway 486/33) required 4.1 milliseconds for the bitmap and 0.89 milliseconds for the polygon. Since polygons are much faster to draw, gesturing programs should use this technique when running on slower processors. We acknowledge that these results are highly dependent on the video display hardware, but the test represents the current state of PC technology.

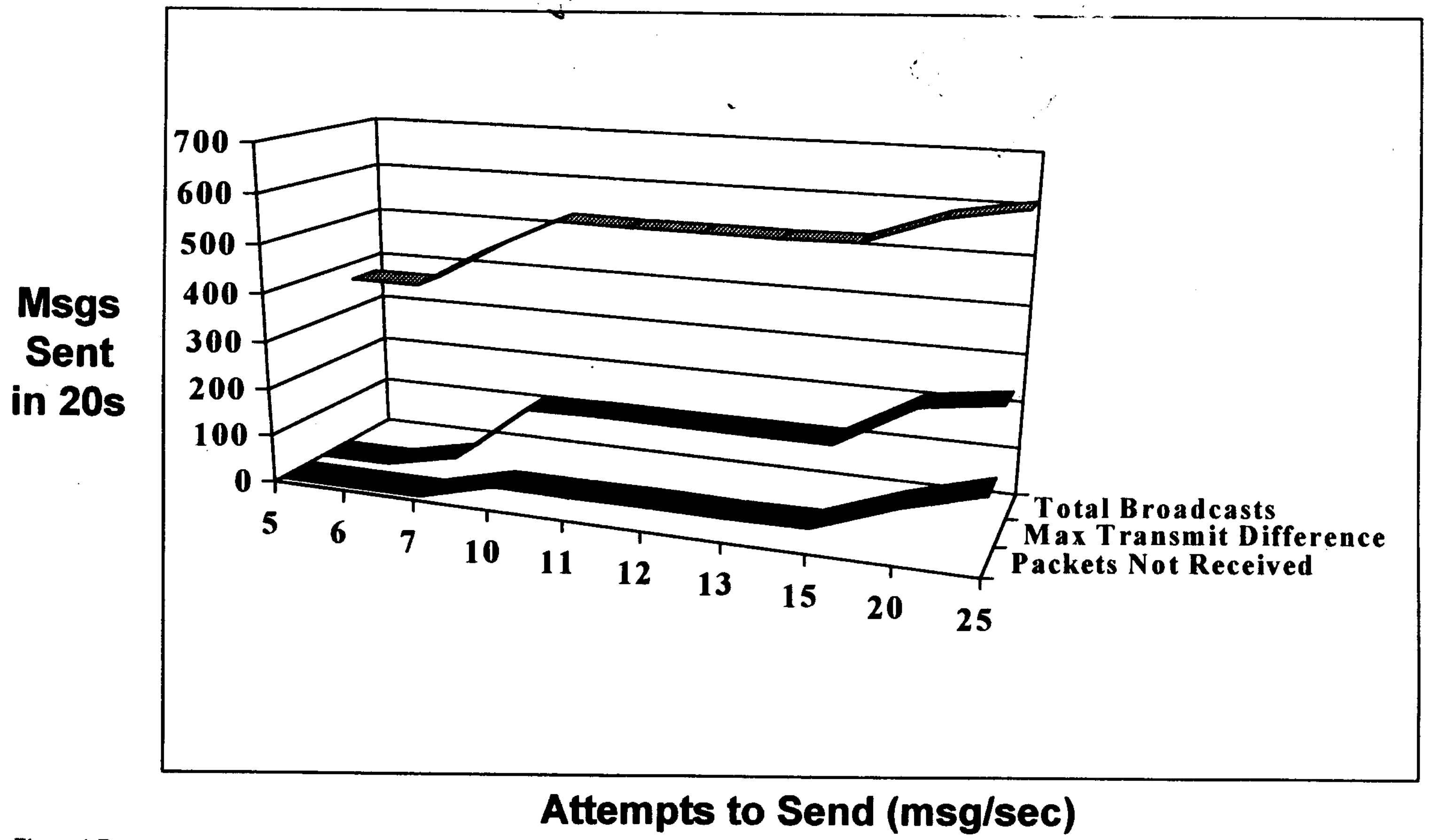

Figure 4. Datagrams Lost Threshold

Table 2 Concurrent Transmitters (messages sent in 20 seconds)

Send Rate	Station A	Station B	Station C	Station D
5	91	91	91	91
6	91	91	91	91
7	88*	121	121	121
10	22**	145	178	179
11	21	139	181	181
12	22	141	181	179
13	22	143	178	179
15	25	138	180	182
20	20	32	261	269
25	22	29	293	265

5.5. Summary

The above tests indicated that unreliable datagram communications are appropriate for gesturing and that a definite limit exists for transmitting and receiving gestures. This limit is determined by processor and network speed. To better understand the processor limits, the following equation was derived from the test data:

$$\text{Utilization: } \rho_i = \beta((n-1)G) + \alpha(G) \leq 100 \text{ percent;}$$

where β = Msg Receive Time + Gesture Redraw Time; α = Msg Send Overhead + Msg Send Time + Application Overhead; G = Gesture rate in msgs per second; n = Number of users gesturing.

Table 3 shows equation results for maximum gesture rates of bitmap and polygon cursors. If a rate of 10 messages per second is required for smooth gesturing, then low-end 386sx/16 systems could handle three concurrent gesturing stations and high-end 486/25 systems could handle ten stations (using bitmap gestures). These calculations assume that no background tasks are active. If large groups are to be supported on PC-based systems, gesturing needs to be viewed as a separate operation from drawing and other schemes such as "point and jitter" need to be explored. It is not required that all stations' cursors be displayed at all times, but based on previous research and our own experience, the richness of the interaction is significantly reduced. Whether this becomes an impediment to successful group interaction is an empirical question.

6. Conclusions and Future Directions

WE HAVE ADDRESSED SOME OF THE PRACTICAL CONSIDERATIONS for gesturing within group support systems applications. We have outlined a framework for implementing gesturing that spans low to high network bandwidth and processing power. The framework also addresses issues relevant to group size and interaction style.

We have argued that the appearance of the gesture cursor should vary based on group size, user activity, and screen size. For small groups the cursor should be large enough

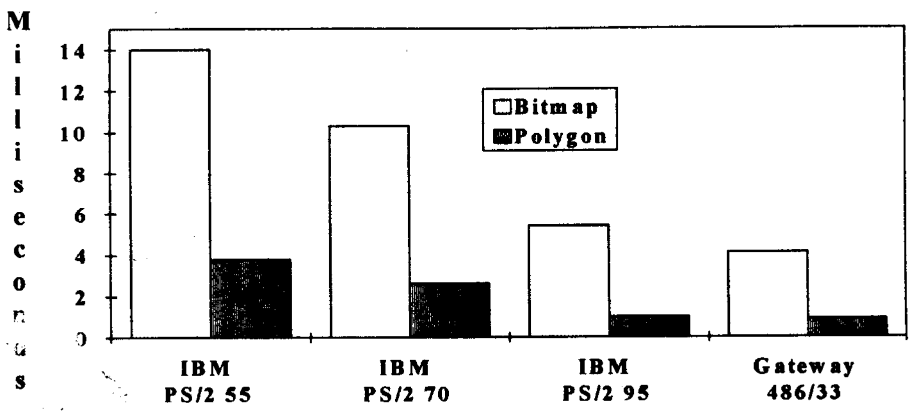

Figure 5. Video Performance

Table 3 Maximum Gesture Rates

	386SX/16			486/25	
n	Bitmap	Polygon	*n*	Bitmap	Polygon
1	38	38	1	100	100
2	18	21	2	54	71
3	11	15	3	37	55
4	8	11	4	28	45
5	6	9	5	22	38
6	5	8	6	19	33
7	4	7	7	16	29
8	4	6	8	14	26
9	3	5	9	12	23
10	3	4	10	11	21

to find and have some sort of owner identification (name, initials, etc.). For large groups (6+) and/or small screens (14″ and smaller), cursors should be made smaller (16×16) and inactive cursors should decay or disappear. Ideally, the cursor shape will provide information about the current actions of the user (drawing, pointing, or requesting the floor). Finally, cursor update modes can be varied to accommodate system limitations. Full motion gesturing should be used whenever possible, with limited motion or point and quiver techniques reserved for slower networks.

During performance tests on PC networks we discovered that there is a large penalty for operating in the Windows GUI environment when transmitting messages. We also determined that the Ethernet or CSMA/CD protocol is somewhat superior in small groups than the Token-Ring protocol using full motion gesturing; either provides more than enough bandwidth. As one might expect, the faster processors were able to transmit and receive more gesture messages per second. More importantly, slower CPUs will spend most of their time processing incoming gesture messages and will not be able to transmit messages of their own. Therefore, if equal participation is required, one must impose an adaptable governing mechanism that limits the transmission rates based on the number of stations participating or else limit the communication via the use of point and jitter and some smoothing mechanisms.

It might be argued that all these problems can be solved by using high-performance workstations. However, for groupware to become commonly used in commercial environments, problems of this sort need to be solved. Further, when groupware applications are distributed geographically relying on low-speed networks, processor problems will be replaced with transmission latency problems. Perhaps the solutions we suggest to address slow processors can be applied to slow networks as well.

NOTE

1. ᐧicrosoft Windows is a trademark of the Microsoft Corporation.

REFERENCES

1. Bier, E.A., and Freeman, S. MMM: a user interface architecture for shared editors on a single screen. In *Proceedings of the Fourth Annual Symposium on User Interface Software and Technology (UIST '91)*, pp. 79–86. South Carolina: ACM Press, November 11–13, 1991.

2. Bly, S.A., and Minneman, S.L. Commune: a shared drawing surface. In *Proceedings of the Conference on Office Information Systems*, April 1990, pp. 184–192.

3. Dennis, A.; Valacich, J.S.; and Nunamaker, J.R. An experimental investigation of the effects of group size in an electronic meeting environment. *IEEE Transactions on Systems, Man, and Cybernetics*, 20, 5 (1990), 1049–1057.

4. DeSanctis, G., and Gallupe, B.R. A foundation for the study of group decision support systems. *Management Science*, 33, 5 (May 1987): 589–609.

5. Dourish, P. Awareness and coordination in shared workspaces. In *Proceedings of the Conference on Computer Supported Collaborative Work*, November 1992, pp. 75–84.

6. Egido, C. Video conferencing as a technology to support group work: a review of its failures. In *Proceedings of the Conference on Computer Supported Collaborative Work*, September 1988, pp. 13–24.

7. Ellis, C.A.; Gibbs, S.J.; and Rein, G.L. Groupware: some issues and experiences. *Communications of the ACM*, 34, 1 (1991): 38–58.

8. Farallon. *Timbuktu User's Guide*. 1988.

9. Greenberg, S. Sharing views and interactions with single-user applications. In *Proceedings of the ACM/IEEE Conference on Office Information Systems*, April 1990, pp. 227–237.

10. Greenberg, S. Computer supported cooperative work and groupware: an introduction to the special edition. *International Journal of Man–Machine Studies*, 34, 2 (1991): 133–143.

11. Greenberg, S., and Bohnet, R. GroupSketch: a multi-user sketchpad for geographically distributed groups. In *Proceedings of Graphics Interface*, June 1991, pp. 207–215.

12. Greenberg, S.; Roseman, M.; Webster, D.; and Bohnet, R. Issues and experiences designing and implementing two group drawing tools. In *Proceedings of Hawaii International Conference on System Sciences*, vol. 4, pp. 320–331. IEEE Press, January 1991.

13. Hayne, S.C. Computer support for graphical brainstorming: a group support system. Working Paper #WPS-10–92, Department of MIS, University of Calgary, 1992.

14. Hayne, S.C., and Pendergast, M.O. Techniques and experiences with object oriented windows-based group support software development. Working Paper, Department of MIS, University of Calgary, 1992.

15. Ishii, H., and Miyake, N. Toward an open shared workspace: computer and video fusion approach of TeamWorkStation. *Communications of the ACM*, 34, 12 (1991): 36–54.

16. Kobayashi, M., and Ishii, H. ClearBoard: a novel shared drawing medium that supports gaze awareness in remote collaboration. *IEICE Transactions on Communications*, 76, 6 (1993): 609–624.

17. Minneman, S.L., and Bly, S.A. Managing à trois: a study of a multi-user drawing tool in

distributed design work. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, New Orleans, April 28–May 2, 1991, pp. 217–224.

18. Minneman, S.L., and Bly, S.A. Experiences in the development of a multi-user drawing tool. In *The 3rd Guelph Syposium on Computer Mediated Communication*, Guelph, Ontario, May 15–17, 1990, pp. 154–167.

19. Nunamaker, J.F.; Dennis, A.; Valacich, J.; Vogel, D.; and George, J. Electronic meeting systems to support group work. *Communications of the ACM*, 34, 7 (1991): 40–61.

20. Olson, J.S.; Olson, M.H.; Mack, L.A.; and Wellner, P. Concurrent editing: the group's interface. In *Human–Computer Interaction: INTERACT'90*. New York: Elsevier, 1990.

21. Pendergast, M.O. Multicast channels for collaborative applications: design and performance evaluation. *ACM Computer Communications Review*, 23, 2 (April 1993).

22. Pendergast, M., and Hayne, S.C.. Alleviating convergence problems in group support systems: the shared context approach. Working Paper, Department of MIS, University of Calgary, 1992.

23. *Random House College Dictionary*. New York: Random House, 1984.

24. Rein, G.L., and Ellis, C.A. rIBIS: a real-time group hypertext system. *International Journal of Man–Machine Studies*, 34, 3 (1991): 349–368.

25. Smith, R.B.; O'Shea, T.; O'Malley, C.; Scanlon, E.; and Taylor, J. Preliminary experiences with a distributed, multi-media, problem environment. In *Proceedings of the 1st European Conference on Computer Supported Cooperative Work (EC-CSCW '89)*, Gatwick, UK, September 13–15, 1989.

26. Stefik, M.; Bobrow, D.G.; Foster, G.; Lanning, S.; and Tartar, D. WYSIWIS revisited: early experiences with multiuser interfaces. *ACM Transactions on Office Information Systems*, 5, 2 (April 1987): 147–186.

27. Stefik, M.; Foster, G.; Bobrow, D.G.; Kahn, K.; Lanning, S.; and Suchman, L. Beyond the chalkboard: computer support for collaboration and problem solving in meetings. *Communications of the ACM*, 30, 1 (January 1987): 32–47.

28. Tang, J.C. Listing, drawing and gesturing in design: a study of the use of shared workspaces by design teams. Ph.d. dissertation, Stanford University, April 1989. Also available as research report SSL–89–3, Xerox Palo Alto Research Center, Palo Alto, CA.

29. Tang, J.C. Findings from observational studies of collaborative work. *International Journal on Man–Machine Studies*, 34, 2 (February 1991): 143–160.

30. Tang, J.C., and Minneman, S.L. Videodraw: a video interface for collaborative drawing. *ACM Transactions on Information Systems*, 9, 2 (April 1991): 170–184.

31. Tang, J.C., and Minneman, S.L. Videowhiteboard: video shadows to support remote collaboration. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, April 28–May 2 1991, pp. 315–322.

32. Group Technologies. Aspects: the first simultaneous conference software for the Macintosh. User's manual, 1991.

33. Valacich, J.S.; Dennis, A.; and Nunamaker, J.F. Group size and anonymity effects on computer-mediated idea generation. *Small Group Research*, 23, 1 (1992): 49–60.

34. Wilson, B. WSCRAWL 2.0: a shared whiteboard based on X-windows. In S. Greenberg, S. Hayne, and R. Rada (eds.), *Groupware for Drawing and Writing*. New York: McGraw-Hill, forthcoming.