PointRight: Pointer/Keyboard Redirection for Interactive Workspaces

Brad Johanson, Greg Hutchins, Terry Winograd

Stanford University
Gates 3B-376
Serra Mall
Stanford, CA 94305-9035
{bjohanso,gmh,twinograd}@graphics.Stanford.edu

Abstract. The ubiquitous computing rooms and interactive workspaces currently being researched and deployed typically have several large screens and dozens of machines which can display to them. Providing convenient and intuitive pointer and keyboard access in such spaces is a challenge. The room should function as a large virtual desktop, and input should automatically be routed to whichever machine is currently displaying to a screen. Unfortunately, existing software and pointer redirection systems are limited to controlling a few dedicated machines with a single keyboard and mouse. PointRight provides intuitive, automatic pointer/keyboard redirection for interactive workspaces. It supports flexible topologies, dynamic changes in machine and screen state, multi-screen machines and can be installed on both permanent and transient machines in an interactive workspace. Versions of the system have been continuously deployed in our prototype interactive workspace over the last 18 months, and a public version is planned for September 2001.

1 Introduction

An interactive workspace is any space where people come together to work collaboratively on some problem with assistance from computational resources that are either built into the environment and/or brought with them in the form of laptops or PDAs. In [1] and [4] we present more details about such spaces and the systems infrastructure required to support them.

Most of these spaces have the characteristic that there are many large displays in the environment. Our prototype interactive workspace, the iRoom, in particular has three 6' diagonal touch-sensitive SMART Boards [8] along a side-wall, a bottom-projected 5' diagonal table-top display, and a custom high-resolution tiled front display. Fig. 1 shows the screen layout. Most of the machines in the iRoom run standard Windows operating systems, so their normal control mechanism is mouse and keyboard—providing flexible, intuitive control of all of the machines to the user in this environment is challenging.

Imagine the following scenario. Alice walks into the iRoom with her laptop to meet Bob who has already arrived. Bob uses a wireless mouse and keyboard that are

permanently in the iRoom to bring up web pages on each of the three SMART Boards. The mouse he is using controls a single pointer that moves across the screens as if they are a single virtual desktop, despite the fact that each screen is being driven by a different machine. Keyboard control is routed to the same machine that the pointer is currently on. Alice decides she would like to load a different web page, so she hits a hot-key on her laptop, and now she can control the pointer across the screens in the space just as Bob was able to do with the wireless mouse. Bob asks to see an application Alice has been working on, so she toggles out of the remote pointer control mode, plugs into a video drop and switches her laptop to display on SmartBoard One. Bob walks up to SmartBoard One and is able to touch interact with the application running on Alice's laptop. When he returns to using the wireless mouse and keyboard, he is able to move the pointer to control Alice's laptop, just as if it were a permanent part of the iRoom infrastructure.

Notice in the scenario that both Alice and Bob redirect pointer control in an intuitive manner. If something is displaying on a touch-screen, the user can interact with it without worrying about which machine is receiving the physical pointer input from the screen. Pointers move across visible screens as if they were a large virtual desktop. When it is non-intuitive that a pointing device, for example the laptop touchpad, should be able to control the pointer in the space of iRoom screens, the user is required to explicitly toggle the control. The system must also provide for multiple machines attached to one screen, and pointer control from multiple different sources.

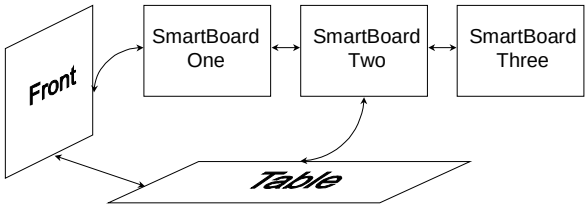

Fig. 1. iRoom Screen Topology and Desired Pointer Transitions

The PointRight system provides for exactly this type of control. It is a flexible pointer/keyboard re-routing system designed primarily for usage in interactive workspaces. Versions of the system have been in use for a year and a half in the iRoom, and have proven indispensable. We are currently packaging a version which we will be distributing as Open Source.

Our contributions are a specification of the design requirements for a pointer/keyboard redirection system for interactive workspaces based on observations of how users perceive the space, and a working implementation of such a system. The key differences from prior work are the support of arbitrary topologies of rectangular screens, multiple simultaneously active pointers, and allowing dynamic changes in mappings between machines and screens.

2 Design Considerations and Requirements

The design of a pointer/keyboard redirection system for an interactive workspace must start with the user. The system should provide control in a manner that is as natural and transparent as possible. It also must accommodate the diverse screen topologies that are possible in an interactive workspace, as well as dynamically changing video connections between screens and machines.

2.1 Types of Input

To begin with, we observe that users in an interactive workspace will be confronted with three basic types of pointer devices, and will have certain expectations as to how they will function:

- **Bound to a Screen:** Some input devices will be inherently associated with a screen regardless of the machine displaying on it. For these devices, users will always expect to control what is displayed on the associated screen. For example, a user will always expect to be able to use a touch-screen to interact with whatever is displaying on the touch-screen. Any pointer re-direction system therefore needs to re-route the touch input to the machine currently displaying on the machine.
- **Bound to a Machine:** In some cases, a user will associate a pointing device with a machine, regardless of the screen to which it is displaying. In this case, the user will expect the control to go to the machine. For example, a laptop user will expect the built-in pointing device to control the laptop, even when the video is being routed to a projector instead of the built-in display. While users may want to re-route the control from the local pointing device and keyboard to the machines displaying on screens in the workspace, they probably want to do so explicitly, and will need a reminder on the local screen to indicate they are currently being re-routed.
- **Bound to Screens in Workspace:** Some devices, like a wireless mouse and keyboard sitting on the table, are not obviously bound to either a screen or a machine, and may be assumed to be bound to the screens in the workspace as a whole. The user may expect that moving the mouse will move a pointer on one of the screens, and that moving off the edge of a screen will move to the adjacent screen. This smooth re-routing of control should happen independently of the machines currently displaying to the screens. If a screen or machine is off, the cursor should never move to that screen. From a pointer movement perspective, controlling a single machine displaying to multiple screens should work the same as controlling multiple machines displaying to the same screens. Keypresses should go to where the pointer is currently located.

In addition to their expectations of how the different classes of pointers work, users will expect all input devices to be simultaneously active. For example, a laptop user should be able to control the pointer on some remote screen in the space at the same time another user is using a wireless mouse to control the pointer on some different

screen. Users would in fact like to be able to control separate pointers on the same screen, but due to limitations in current operating systems this is difficult or impossible to provide.

2.2 Flexible and Dynamic Topology

Interactive workspaces will have a variety of topologies. Not only can screens tile a plane in arbitrary ways, but there may be folds and corners. For example, in our iRoom (Fig. 1) the left edge of the table display connects to the bottom edge of our front display. A pointer/keyboard re-routing system therefore needs to support arbitrary 3D manifolds composed of rectangular screens of different sizes and aspect ratios, providing smooth motion across all of them.

The topology may also be dynamic. There will be permanent screens, some of which may switch between displaying several different machines. Some machines may be able to output video to several different screens simultaneously. Some screens may be on rollers, and laptops may enter and exit the workspace. In addition to maintaining the state of the room topology, any pointer redirection system must also track the changes in the on/off state of machines and screens, and the mapping between machines and screens.

3 Implementation

PointRight operates in either server or client mode. In *server* mode, PointRight accepts and redirects mouse and keyboard events from local input devices, and in *client* mode, it adjusts the local cursor position and generates keystrokes based on pointer and keyboard events from remote PointRight applications operating in server mode. To participate in PointRight, a machine must be running the PointRight application and be connected through the local network to the interactive workspace infrastructure.

The PointRight system currently works with Windows 9x/NT/2000 and Linux, although only server mode currently works under Linux. Right now the system is specific to the iRoom, but we are working on a general version that should be easy to set up and configure for any ubiquitous computing room and should be available for download by September, 2001.

3.1 Room Topology Database

PointRight stores the topology of the room as a collection of *screen*, *machine*, and *connection* objects. Screens are video sinks, and machines are video sources. Connections specify a valid pointer transition between a section of an edge on one machine and a section of an edge on another machine. Screen objects have two pieces of state associated with them: their on/off status, and the machine currently being displayed on them. Machine objects store the on/off state of the physical machine they represent (or equivalently whether the PointRight application is currently running on

that machine). Fig. 2 gives an example screen topology that demonstrates the flexibility of this specification framework:

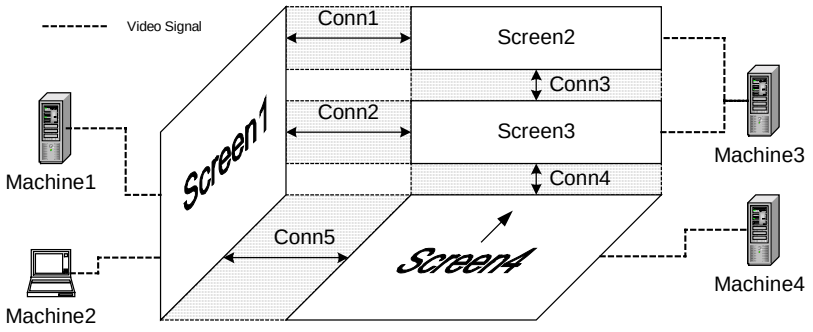

Fig. 2. An Example Topology with Screens, Machines and Connections. Note that Screen1 has two video sources, Conn1 and Conn2 connect to the same edge of Screen1, Conn5 connects a left edge to a bottom edge, and Machine3's desktop displays across Screen2 and Screen3

The Event Heap [4] is used to propagate state change events from physical machines and screens to their representative objects stored in each PointRight application. These events are posted and retrieved to and from an Event Heap server machine, and persist for a specified period of time. PointRight applications post machine on/off events when opened or closed, and, in the iRoom, screen state events are posted by projector controller applications which control physical projector state over a serial line. Events persist for two minutes, and all active machines post events refreshing their current state every two minutes. If a PointRight application does not see an event from a previously active machine or screen, the state of that object in the local database is updated to "off." Using this soft-state mechanism allows for the graceful handling of crashes, provides a simple mechanism for applications that are starting up to retrieve the current state of the topology, and insures that applications always have an accurate model of the state of the screens and machines in the topology to within two minutes.

3.2 The PointRight Application

PointRight consists of a single application that is run on any machine that will be either a source or a target for pointer and keyboard events. Each machine can act as either a client and receive events, or as a server sending events to remote machines (in the remainder we use client or server to mean a PointRight application running in one of those modes). Due to limitations in the event queue model of most operating systems, PointRight does not allow a remote mouse and keyboard to interact with local applications while the local mouse and keyboard are controlling another machine. In other words, the PointRight application cannot act as both server and client simultaneously. As mentioned in the previous section, the programs depend on

access to the Event Heap to maintain server state. All pointer and keyboard control events are sent over direct socket connections to the appropriate client.

The client mode is the more straightforward of the two. The client knows the area of the screen for which it is responsible, for example the second head of dual-head system may be responsible for x=1024-2048 and y=0-768. The value passed from the server is in a normalized range for x and y. These normalized values are scaled to the x and y range managed by the client, and the pointer on the client machine is set to this position. Keypresses are simply inserted into the event queue on the local machine.

Each server stores the topology as described in the previous section. In addition, it maintains three pieces of state: the current *screen* to which it is re-routing pointer control, the current *machine* to which it is re-routing pointer control, and the current position on that screen. If the server machine has a relative pointing device, as edges are reached it looks up the adjacent screen, if any, and begins re-routing to the machine attached to that screen. If the adjacent screen or machine is off, it finds the next available screen in the direction the pointer exited the screen, or clips the cursor to the current screen if no suitable screen is found. Anytime the machine to which events are supposed to be re-routed is the machine running the server, PointRight automatically switches back to client mode, but continues monitoring the cursor in case it goes back off an edge and needs to be re-routed again (in which case it re-enters server mode). One special case is when the input device for the machine is a touch-screen or other direct input device. In this case PointRight continuously monitors the screen associated with the touch-screen and routes control to whichever machine is displaying on the touch-screen.

The PointRight application can be switched from client to server mode. In client mode, the local machine can be used normally, or remote machines can control it via pointer/keyboard redirection. Server mode, can either be toggled into manually by the user via hot-key, or automatically when the system determines control needs to be re-routed. When a machine not physically connected to a touch-screen begins to display to it, the directly connected machine automatically switches to server mode and begins re-routing the touch-screen input to the newly connected machine. The mode is also auto-switched when a mouse connected to a machine displaying to one of the screens that is part of the topology is used. In this case when the edge of the screen is reached, server mode is entered and control is automatically routed to the adjacent screen. This happens most typically when a laptop is being displayed to a screen and the pointing device built into the laptop is moved to the edge of the laptop display. In all cases, the video signal output by a machine is grayed out during pointer control to provide the user a cue that control cannot be routed to that machine (it can't be a client and server simultaneously).

4 Related Work

The x2x [11] and x2vnc [2] systems allow you to configure your X-Windows machine such that mouse and keyboard control are redirected to either another X-Windows machine or a Windows machine, respectively, when the mouse reaches the edge of the

desktop on the controlling machine. They are similar to our system, but don't support arbitrary topologies, allow multiple machines switched to a single screen, or provide for multiple simultaneous redirections.

To the user, the feel of PointRight is very much like the support for multi-screen desktops available for Apple, X-Windows and MS Windows machines. However, these assume a single machine with multiple displays, not the many to many mapping supported by PointRight.

VNC [7] and similar software products allow keyboard and mouse re-routing from one machine to another along with mirroring the remote display. They are designed primarily for remote machine usage, and don't provide the smooth mapping across multiple machines and displays we achieve with PointRight.

The Pebbles system [5] and other CSCW systems allow multiple-cursors (controlled by PDAs in the case of Pebbles) simultaneously on a single machine, but don't support transferring control between many adjacent screens. Our system specifically doesn't address the issue of multiple people interacting on one machine, but could be combined with such a system.

Rekimoto in [6] has shown an augmented desktop where a laptop mouse can drag items on and off the laptop display to a top projected table display. This domain of having control over large amounts of screen space controlled by different machines is similar to ours, but his system is focused on exploring interaction modalities rather than trying to provide a general mechanism for keyboard and pointer re-direction.

5 Conclusions

One of the first things we found upon completion of the iRoom was that we needed an intuitive way to provide users with mouse and keyboard control for a large collection of screens. PointRight has evolved through two generations to fill that role, and has proved invaluable over the last year and a half.

Unlike the other systems we know of, it provides for large collections of screens and flexible layout topologies. It supports dynamic environments, multiple machines per screen, and multiple screens per machine, and allows for direct interaction input devices like touch-screens. It requires only that an Event Heap be running in the interactive workspace, and that participating machines install a small application. Finally, and most importantly, the system works the way users expect.

We feel the system is key for any space with a large number of screens and machines, even if it is only to allow easy debugging of more complex monolithic applications being designed to run in the space. We are currently working on a software release we intend to make publicly available, and are exploring the possibility of dynamic addition and removal of screens, machines and connections to and from the topology.

Finally, the fluidity of using the PointRight system is difficult to convey without seeing it in action. We have prepared a video of the system for the reader in streaming RealVideo format which may be accessed at: http://graphics.stanford.edu/papers/pointright-ubicomp/.

Acknowledgments

The Interactive Workspaces project is the result of efforts by many students; see [3] for an exhaustive list and more complete project information. Thanks to Bryn Forbes, and Rito Trevino in particular, who helped figure out how to tap into the Windows and Linux mouse and keyboard event systems. Thanks to Susan Shepard for her help making the video and keeping the iRoom stable enough for development. Special thanks to Maureen Stone for her helpful criticism on early drafts. The work described here is supported by DoE grant B504665, and by donations of equipment and software from Intel Corp., InFocus, IBM Corp. and Microsoft Corp.

References

[1] Fox, A., Johanson, B., Hanrahan, P., Winograd, T., "PDAs in Interactive Workspaces," Computer Graphics and Animation, May, 2000.

[2] Fredrik Hubinette, "x2vnc," http://fredrik.hubbe.net/x2vnc.html.

[3] Interactive Workspaces Project at Stanford University, http://graphics.stanford.edu/projects/iwork. See also http://graphics.stanford.edu/projects/iwork/pointright.

[4] Johanson, B., Fox, A., "The Event Heap: A Coordination Infrastructure for Interactive Workspaces", submitted to UbiComp 2001, Atlanta, GA, USA, 2001. (reviewer copy only available at: https://graphics.stanford.edu/papers/eheap2/).

[5] Brad A. Myers, Herb Stiel, and Robert Gargiulo. "Collaboration Using Multiple PDAs Connected to a PC." Proceedings CSCW'98: ACM Conference on Computer-Supported Cooperative Work, November 14-18, 1998, Seattle, WA. pp. 285-294.

[6] Rekimoto, J. "Augmented Surfaces: A Spatially Continuous Work Space for Hybrid Computing Environments." CHI'99, pp. 378-385.

[7] Richardson, Tristan, Quentin Stafford-Fraser, Kenneth R. Wood & Andy Hopper, "Virtual Network Computing", *IEEE Internet Computing*, Vol.2 No.1, Jan/Feb 1998 pp33-38.

[8] Smart Technologies SMART Board, http://www.smarttech.com/smartboard/.

[9] Weiser, M., The computer for the twenty-first century. *Scientific Americ*an, pages 94–100, September 1991.

[10] P. Wyckoff, S. W. McLaughry, T. J. Lehman and D. A. Ford. TSpaces. IBM Systems Journal 37(3). Also available at http://www.almaden.ibm.com/cs/TSpaces.

[11] "x2x," http://ftp.digital.com/pub/DEC/SRC/x2x/.