Advanced Multi-Display Configuration and Connectivity

by

Ritchie Argue

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2007

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "Advanced Multi-Display Configuration and Connectivity" by Ritchie Argue in partial fulfillment of the requirements for the degree of Master of Computer Science.

Dated: August 7, 2007

Supervisors:

Dr. Kori Inkpen

Rev. Prof. Dr. Dr. Kellogg S. Booth

Reader:

Dr. Stephen Brooks

DALHOUSIE UNIVERSITY

DATE: August 7, 2007

AUTHOR: Ritchie Argue

TITLE: Advanced Multi-Display Configuration and Connectivity

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: M.C.Sc. CONVOCATION: October YEAR: 2007

In memory of Harold and Earla Hyde

Table of Contents

List of Tables

List of Figures

Abstract

Adding multiple displays to a personal computer is a popular way to manage the ever-increasing amount of information users are faced with, and to share information with others. However, the configuration and use of multiple displays has long been bound by technical limitations which only enable a subset of imagined possibilities. This thesis attempts to address three of the primary limitations: secondary displays may only be added as an extension of the desktop or as a duplicate of an existing display, they must be directly attached to a computer, and they can only be used by a single user at any one time. A binary patch was developed to augment the display configuration capabilities of Mac OS X. This allows a secondary display to be placed anywhere in the logical screen space—partially overlapped, completely disjoint, or inset—rather than just abutting (extended desktop) or duplicating (mirror/clone mode) an existing display. A suite of applications and drivers was then developed to allow a co-located networked computer to transparently act as a secondary display for other computers, as though it were directly connected via a standard DVI or VGA cable. It was found that by exploiting some of the properties of a modern window server, this network-attached display could act as a communal interactive surface. Such a configuration immediately allows users to gain parallel-task groupware benefits from existing single-user applications without having to write specialized programs that rely on a groupware toolkit. This thesis provides the technical framework on which further multi-display research may be built.

List of Abbreviations Used

AGP	Accelerated Graphics Port
API	Application Programming Interface
CPU	Central Processing Unit
CRT	Cathode Ray Tube
DVI	Digital Visual Interface
EDID	Extended Display Identification Data
GPU	Graphics Processing Unit
GUI	Graphical User Interface
KVM	Keyboard/Video/Mouse switch
LAN	Local Area Network
LCD	Liquid Crystal Display

MDE Multi-Display Environment

MDG Multi-Display Groupware

OS Operating System

PC Personal Computer

PCI Peripheral Component Interconnect

QOS Quality Of Service

SDE Satellite Display Emulator

SDG Single-Display Groupware

TV Television

VGA Video Graphics Array

VNC Virtual Network Computing

VRAM Video RAM

Glossary

accelerated graphics port — An internal bus format for attaching graphics cards to a **PC**.

application — In the context of computer software, a program running in **user space**, or outside the system **kernel**. Most software that users interact with falls into this category.

application programming interface — An interface specification that a software library provides to allow programs to utilize the services the library offers.

cathode ray tube — The original analogue technology for dynamic electronic image display. Slowly being replaced by newer digital technologies such as **LCD**, **DLP**, and plasma displays.

central processing unit — The primary general-purpose data processing component in a **PC**. Contrast this with a **GPU** which is largely dedicated to operating on graphics data.

client — In the context of an **SDE**, a client provides graphics data to a **display** device.

clone mode — See **mirror mode**.

co-located — See **co-present**.

co-present — Or **co-located**. Located in the same environment. For example, two users are co-present if they are working in the same room.

co-present groupware — **Groupware** intended to support **co-present** users.

desktop — A region describing the logical area a **screen** (or screens) occupies in **global screen space**.

digital light processing — DLP is the marketing term for what is generically known as a digital micromirror device. These chips utilize an array of microscopic electromechanically actuated mirrors to form an image with reflected light.

digital visual interface — An analogue and digital video interface standard. See also **VGA**.

display — A hardware output device that provides a dynamic two-dimensional image with which to communicate with a user. Examples include **CRT**, **LCD**, or plasma direct view **monitor**s and **LCD** or **DLP** front projection **display**s.

extended desktop — A multi-display configuration in which two or more physical **display**s are represented as logically adjoined but mutually exclusive in **global screen space**, providing the user with a larger **desktop** than can be accomplished with a single display. See also **mirror mode**.

extended display identification data — Data sent from a **display** (**sink**) to a **PC** (**source**) identifying the operating characteristics of the display, such as a string representing the display name, and a set of display timings.

framebuffer — A region in memory representing the current image being shown on a given **display**.

framework — On Mac OS X, a container that encapsulates shared resources such as dynamically-linked shared libraries, header files, and documentation.

global screen space — A two dimensional conceptual display plane in which **desktop**s are defined. Traditionally, global screen space may contain multiple desktops, some overlapping (**mirror mode**) and some abutted (**extended desktop**).

graphical user interface — A user interface that uses images in addition to text to convey information to a user.

graphics processing unit — A dedicated **PC** component that performs graphics operations and frees the **CPU** from this task.

groupware — Computer based system that supports groups of people engaged in a common task (or goal) and that provides an interface to a shared environment.

intelligent satellite — A computer terminal with more graphics processing capabilities than traditionally available. This allowed graphics processing operations to be offloaded from the central mainframe, allowing more **CPU** time to be spent on other computations.

kernel

The central component of an operating system. A kernel manages system resources and the communication between hardware and software. In addition, a kernel provides an abstraction of the hardware (**API**) to application software. The kernel used by Mac OS X is a monolithic kernel, running in its own protected address space. Hardware drivers may also run in this space as kernel extensions.

kernel extension

Software that runs in the same address space as the **kernel**. Typically used for drivers that need low-level access to hardware. Because they run alongside the kernel, instability in a kernel extension may affect the kernel as well.

keyboard/video/mouse switch

A hardware device that switches a single set of inputs (typically keyboard, mouse) and output (video) between multiple **PC**s.

liquid crystal display

A flat panel **display** technology. For computer use, first made popular on laptops and now used for most **PC** displays, replacing **CRT**s.

local area network

An IP-based **network** that provides interconnectivity between local computers.

local display

A **display** that is directly connected to a user's **PC** via a point-to-point interface, such as a **DVI** or a **VGA** cable.

mirror mode

Or **clone mode**. A display mode in which two **display**s share the same **framebuffer**, in effect displaying the same **desktop**. See also **extended desktop**.

monitor — A **display** based on a direct-view technology such as CRT, LCD, or plasma.

multi-display — A configuration in which multiple **display**s are connected to a single PC.

multi-display environment — A configuration in which more than one **display** is visible at a time and there is some computational coordination between them. For the purposes of this research, this may be as simple as a single PC with two displays, such as a laptop connected to an external display.

multi-display groupware — **Groupware** intended to support **co-present** users via two or more **display**s.

network — For the purposes of this research, an IP based data transport layer. Examples include Ethernet and 802.11x wireless.

operating system — The underlying software that manages the hardware and software resources of a computer.

peripheral component interconnect — An internal bus format for attaching peripherals to a PC.

personal computer

A small, lightweight computer intended to support the needs of a single user. For the purposes of this research, the term PC will be used to describe a machine running either Mac OS X or Microsoft Windows. If necessary, the operating system will be specified, such as 'a *Windows* PC.'

product

For the purposes of this research, a product is software either freely or commercially available. Contrast this to a software **project**, which is typically inaccessible.

project

For the purposes of this research, a project is software developed in a research context. This software is typically not available for use outside of the lab in which it was developed. Contrast this to a software **product**, which is either freely or commercially available.

quality of service

The guarantee of an upper bound on the time it takes a packet to traverse a **network**. QOS streams are typically used for real-time applications such as voice calls, where packet loss or lag is detrimental. This is in contrast to other data that may be on a network, such as that used by a bulk file transfer.

remote desktop

A configuration in which a local **PC** acts as both **display** and input surface for a distant PC connected via a **network**. Input is forwarded to the distant PC, and graphics information is received back. Typically used for telecommuting and remote administration.

remote display

For the purposes of this research, a **display** that is connected to a **PC** via an interface that supports addressability and routing, such as a **network**.

satellite display — For the purposes of this research, a display with more processing capabilities than traditionally available. This allows graphics operations to be offloaded from a **PC** to the display, allowing for more flexible connectivity and compositing effects. See also **intelligent satellite**.

satellite display emulator — A prototype software implementation used to illustrate the features and behaviours of a **satellite display**.

screen — The logical, or internal-to-a-computer, representation of a **display**. This is the high-level equivalent to **framebuffer**.

server — In the context of an **SDE**, a server receives graphics data from a **client PC**, and displays it.

single-display groupware — **Groupware** intended to support **co-present** users via a single **display**.

sink — A device that consumes information being transmitted from a **source**. In the context of an **SDE**, a sink is also known as a server.

source — A device that generates information to be transmitted to a **sink**. In the context of a **SDE**, a source is also known as a client.

television — An integrated signal receiver (tuner) and **display** device. This is becoming a colloquialism for a large shared display in the home.

user space

A program space external to the **kernel**. **Applications** running in this space may only interact with software in the kernel via a rigidly defined set of APIs. Each application runs in its own address space, insulated from other applications. In this way, instability or crashing of one application does not affect other applications or the kernel.

video RAM

A region of memory that supports a **framebuffer**. Historically, VRAM was dual-ported dynamic RAM that could be read by the graphics hardware at the same time as it was being written to. As dynamic RAM speeds have increased and the needs of graphics ram has expanded beyond that of simply supporting a framebuffer, VRAM is becoming a colloquialism to describe any dynamic RAM used by the graphics system.

video graphics array

Typically (and incorrectly) used to refer to the popular 15 pin D-sub analogue **PC** display connector and associated cabling. More properly, **VGA** designates an analogue PC display standard with a resolution of 640 × 480 pixels. See also **DVI**.

virtual network computing

A client-server suite of software that allows a remote PC to be used as though it were locally available, by forwarding input and output data over a **network**.

warp

For the purposes of this research, a cursor may be warped, or instantaneously moved, between two points separated by an arbitrary distance.

wheel of reincarnation

A phenomenon in which functionality is added to a graphics system until it is no longer cost effective, at which point the system is split into multiple low-cost components. As these components are then improved, this process is potentially repeated *ad infinitum*. Defined by Myer and Sutherland in 1968.

window server

A system process that is responsible for managing GUI windows, their layout and their composition in the framebuffer. Also handles display configuration and input event routing.

Acknowledgements

The author wishes to thank Kellogg Booth for urging him to start on this journey, and Kori Inkpen for allowing him to complete it. He also thanks his editors: Jon Salter, Jim Wallace and Kirstie Hawkey. Finally, he is grateful to his family and friends for their unwavering support.

Portions of this work were presented as a poster [7] at the Graphics Interface 2007 conference in Montréal, Quebec.

Chapter 1

Introduction

This thesis focuses on the flexibility and ease of use of current Multi-Display Environments (MDE), and presents two projects to address each of these issues. The first project extends the number of ways in which a user may configure an MDE beyond the familiar extended desktop and mirror modes. The second project allows displays to be indirectly attached to a PC, and simultaneously shared between multiple PCs.

There is no denying the popularity that multi-display systems are currently enjoying, as both business and home users take advantage of inexpensive dual-head graphics cards to add a secondary display to their PCs [29]. There are two immediate differences between single and multi-display systems: multiple displays provide a larger overall workspace as well as the ability to visually partition tasks into logical groupings. The first property helps users manage the glut of information they are now presented with from sources such as the Internet and large digital media collections. Visual partitioning makes it easier to multitask and to deal with many such streams simultaneously [28].

While many users prefer multi-display setups for those reasons, the interaction with this setup is different from single display configurations. For example, global GUI objects such as the Windows task bar or the Mac OS X menu bar only exist on the primary display, making interaction with them difficult if the cursor is on a secondary display. There is much to study regarding the effectiveness of GUI artifacts developed for a single display in a multi-display world, for example, task or menu bar placement and window management operations [37, 69]. This thesis focusses on the low-level issues of configuring and connecting multiple displays.

1.1 Motivation

Users have had access to multi-display personal computers for the past 20 years. In that time, the technology has become substantially cheaper, but its use has not changed significantly: graphics hardware is added to a computer to allow the user to connect a secondary display, which is then

logically added to extend or duplicate the existing desktop area. In the past, expensive and immature supporting technology imposed a set of constraints that have since become entrenched in the current usability of multi-display systems. This thesis considers three such limitations, outlined in Sections 1.1.1–1.1.3, and proposes solutions for each. First, current PC operating systems support the configuration of multiple displays in only two ways: by mirroring an existing display, or by extending the desktop. These choices are a small subset of the total number of configuration possibilities, but it is not possible to evaluate the utility of the remainder without additional tools. Second, the requirement for point-to-point connectivity of graphics interfaces such as Digital Visual Interface (DVI) and VGA makes it difficult to dynamically reconfigure the display topology in a multi-display environment. Adding multiple displays to a device with constrained graphics hardware, such as a laptop, is also problematic. Finally, when attached to a computer running a single-user operating system, displays may only be used by a single user at a time. As displays become larger and more ubiquitous, it may be beneficial to share them for both personal and collaborative work.

1.1.1 Multi-display configuration limitations

In the past, designers and artists using expensive graphics workstations wanted to dedicate the entire surface of a high-performance display to the artwork being created, so it made more sense to place tool palettes, menus, and other GUI widgets on a cheaper, smaller secondary display that could be easily accessed. The obvious solution was to span the desktop space across both displays, and allow the cursor to be intuitively dragged across the boundary to facilitate interaction on both displays. Similarly, business users needed a way to give presentations at meetings via a video projector while still being able to see the presentation content themselves, similar to the way a traditional overhead projector works. To support this, early laptops offered a video output which simply duplicated the contents of the onboard display.

These initial two uses for secondary displays led to the adoption of the two multi-display configuration choices available today: an extended desktop configuration and mirror/clone mode. However, these choices are not representative of the total available options. Figure 1.1 illustrates some of the many other possible relationships between two screens that exist in two dimensions.

Scenario 1 illustrates why the two configuration modes currently provided may not be sufficient for all users.

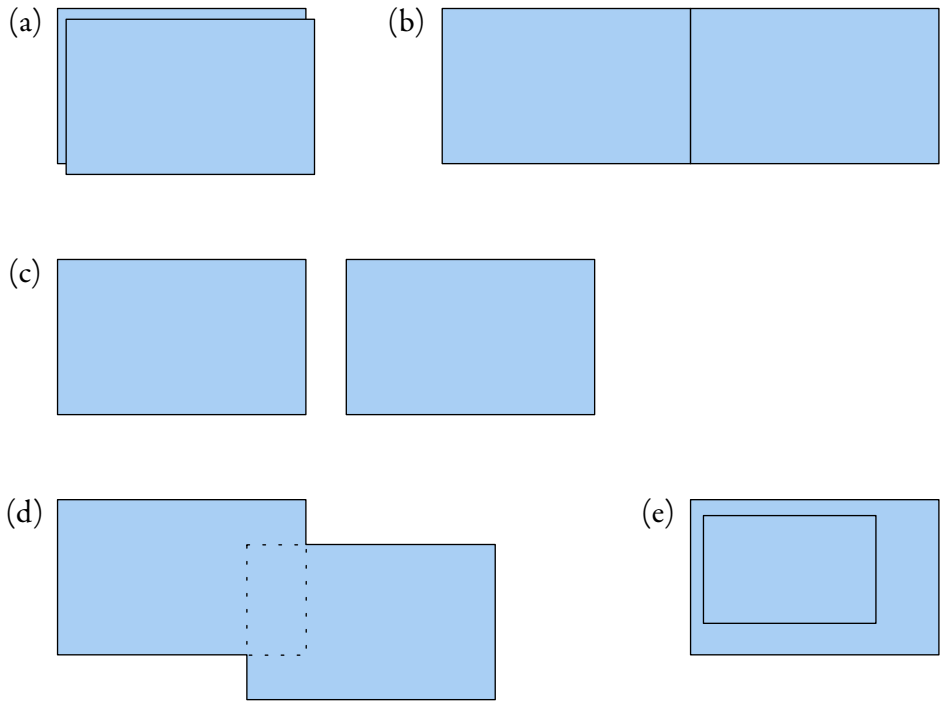

Figure 1.1: Possible relationships between two screens in two dimensions. The familiar mirrored mode is shown in (a), while the arrangement in (b) indicates extended desktop mode. There is a logical gap between screens in (c), and the screens in (d) and (e) have varying degrees of overlap.

Scenario 1

Alice uses a laptop as her primary computer, although it is most often used on a desk both at work and at home. In these locations, it is augmented by peripheral devices such as a keyboard, mouse, and external display. Because it is larger, Alice uses the external display as the primary screen in these situations and uses the built-in LCD panel as a secondary workspace to display notifications such as new email alerts. Alice finds the extended desktop configuration troublesome, as she often overshoots scrollbars or other GUI widgets with her cursor on the primary external display, losing the cursor onto the laptop's built-in LCD. She wishes there was a way to display different information on both screens, as the extended desktop configuration allows, while employing a different mechanism to transfer the cursor between screens. This would allow her to retain the familiar hard-edged behaviour of a single screen, thus preventing her from losing the cursor as easily.

While the diagrams in Figure 1.1 focus on the visual relationship between two screens, this relationship also influences the user interface. We discovered that because the cursor cannot exist outside of a screen, it may not be able to jump a gap between two screens. This would allow different information to be presented on each display similar to the extended desktop mode, but each display would retain the familiar hard-edged behaviour of a single screen.

As graphics hardware has become much more capable, the limitations imposed by the two standard configuration modes are no longer necessary. Removing these constraints may better support variations on the old modes of usage, and provide the fundamentals for new ones.

1.1.2 Physical connectivity limitations

The bandwidth required to transmit video data from a computer to a display is higher than that needed by any other component of a typical computer system. As such, the buses responsible for moving this data have been implemented as point-to-point channels using, for example, the familiar DVI or VGA formats. This one-to-one mapping dictates that displays must be directly plugged into the computers they are intended to be used with, which results in a number of usability conventions that have remained unchallenged until now. These are illustrated in the following scenarios.

One result of the one-to-one mapping between displays and PCs is that the inputs to a set of displays may not be shared, even if the output is. A single user with two displays may consider them as a single workspace or desktop, regardless of the arrangement of different sources driving them.

Scenario 2

> Bob sits at his desk, which houses one each of a Mac OS X and Windows PC, each with its own display. As a web developer, Bob authors content on the Mac and uses browsers on both the Mac and Windows PCs to check cross-platform compatibility. In addition, he runs an email client on the PC and an instant messaging application on the Mac. Because the content authoring application uses a large part of the screen, he would like to move the Mac web browser and instant messaging client to unused space on the PC display, while still being able to see the PC browser and email client.

Another consequence of the one-to-one mapping between displays and PCs is that laptops typically support only a single external display, regardless of the power of the graphics processor they

contain.

Scenario 3

> It's been a good year for the company Alice works at, and there is a surplus in the equipment budget. Corporate IT has determined that using multiple displays offers a productivity boost and decided that everyone working in Alice's office should have two displays. Alice wants to continue using her laptop as it contains all her files and applications, but it can only drive a single external display in addition to the onboard LCD. She would prefer to use both external displays and ignore the built-in LCD, but then she would have to migrate to a regular computer that had a dual-head graphics card.

An equally constraining result is that most displays allow only a single computer to be connected at any one time.

Scenario 4

> A university has installed a rear-projected high resolution tiled display for scientific visualization research. The dedicated computer cluster driving the display handles the tiling and scaling as well as natively running high performance parallel rendering visualization software. The lab technicians would like to quickly and easily be able to connect different PC systems to the display without having to invest in further video processing or switching hardware.

Addressing the problems raised in these scenarios requires changes to the conventions dictated by these one-to-one mappings.

Graphics buses provide a trade-off between flexibility and performance: they move an incredible amount of data between two devices, but offer very little in terms of routing or addressability. Hardware switching systems such as KVMs have been developed, but these are little more than mechanical solutions that simulate a cable being physically moved from one connector to another.

Being able to dynamically route video between all display surfaces in a given environment would give users the flexibility to choose the best display for their content, rather than being limited to whichever display happens to be directly connected to the device they are interacting with

at the time. High-speed networking and aggressive data compression techniques make this a possibility, by routing graphics data over a multi-point network rather than a point-to-point graphics link.

1.1.3 Single user limitations

When running a single-user operating system on a PC that is directly connected to a display, it is clear that the display may only be driven by a single user at a time. But in an environment with many display surfaces and many users, some displays may be shared. This may occur either by design or in the context of an *ad hoc* activity. However, these displays are often only shared in the physical domain: many users may view them simultaneously, but access to their inputs are serially shared.

Scenario 5

> Five co-workers share an open plan office with an adjoining meeting area and a large shared projection screen for giving presentations. Three of them are working on the latest product design and would like to quickly share with one another snapshots of what they have accomplished. They would each like to use the projector as an extended desktop display for their workstations, and they think it would be even better if they could simultaneously display more than one employee's work on it.

The single-user, single-display model of PC usage is being outgrown by the PC's pervasive integration into the lives of its users—lives that interact with other people. Sharing some aspects of a computer, such as input devices, is inherently physically limited. As a broadcast media, displays are easy to share as long as there is enough total real-estate for everyone who wishes to participate. This should be the only limitation, and the underlying technology should not be a concern.

Scenario 6

> Carol and Dave are in their living room, watching TV on a flat panel display and planning a vacation on their laptops. As they find flight or lodging information on the Internet, they wish to compare prices and schedules. They would like to use the TV as a large shared display for information they find to facilitate this.

One solution for both Scenarios 5 and 6 is to replace the traditional point-to-point graphics connection with a multi-point network or bus. Features supported by such a topology can be used to offer new routing options for graphics data, such as broadcasting or funnelling several streams to one device.

1.2 Problem Statement and Research Objective

Current multi-display systems are overly restricted in today's display-rich, collaborative environments. Even in a single-user setting, multi-display systems do not allow for the configurations demanded by some situations. These usability problems are caused by an initial technological limitation; however, even after the technology issues have been solved, outmoded usage continues. The goal of this research is to investigate and extend the capabilities of current multi-display computing environments.

1.3 The Research Objectives

Initial research has revealed the popularity of multi-display systems, regardless of current technological constraints [29]. To better support the use of multi-display systems, I considered the configuration and connectivity limitations discussed in Sections 1.1.1–1.1.3, and proposed solutions for each, which form the objectives for the research.

Improve multi-display configuration: **Reflect**, a tool designed to blur the distinction between the extended desktop and mirrored multi-display modes on Mac OS X, is used to overcome the previously limited configurability of multi-display PCs. I developed two software components to accomplish this: a patch to the kernel to enable the necessary functionality in the OS and a tool to augment the system-provided multi-display configuration utility. I also developed an additional tool called **MouseWarp** which allows the user to warp the cursor betwen two displays that may be inaccessible by the traditional method of dragging.

Remove physical connectivity limitations: To address the one-to-one physical connectivity requirement imposed by graphics interfaces such as DVI and VGA, I designed a new display device architecture. This architecture replaces the traditional point-to-point connection with a multi-point network or bus, logically freeing display devices from being directly connected to a specific

PC. To demonstrate this concept, I developed a new display device, a **satellite display**, and created a prototype implementation for Mac OS X.

Allow multi-user connectivity: In addition to simply emulating a point-to-point link between a display and a computer, the addressability and routing abilities of a multi-point network were also used to add multi-user capabilities to the satellite display project, while allowing it to retain the simplicity of the existing extended desktop mental model.

1.4 Overview of the Thesis

Chapter 2 reviews related literature, including a technical discussion of multi-display systems, remote desktop solutions, and co-located groupware projects. Chapter 3 illustrates the development of the Reflect display configuration utility. The satellite display is presented in two parts. Chapter 4 describes an ideal satellite display design, given unlimited engineering resources. This design is scaled back in Chapter 5, and used as the basis for a prototype implementation of suitable scope for a master's thesis. Lastly, insights gained while designing and implementing both the Reflect and satellite display projects are given in Chapter 6, including an assessment of the progress made on each of the three objectives stated above. Recommendations are made for future areas of study and future system designs, and conclusions are drawn from this work.

Chapter 2

Related Work

A wide variety of projects, products, and concepts influenced this research. These range from the now-standard multi-display single-user configurations to remote desktop utilities and Multi-Display Groupware (MDG) collaboration tools. The overarching theme tying this work together is the notion of the MDE. This encompasses any configuration in which more than one display is visible at a time, and there is some computational coordination between them. This environment may be used by one or more users.

The taxonomy of co-located display environments is typically arranged along two axes: the number of users, and the number of displays (Figure 2.1). While the research described in this thesis focuses on environments with multiple physical displays and any number of users, single-display concepts such as virtual multiple displays and Single-Display Groupware (SDG) are still applicable so features of these will be discussed as they relate to the larger MDE context.

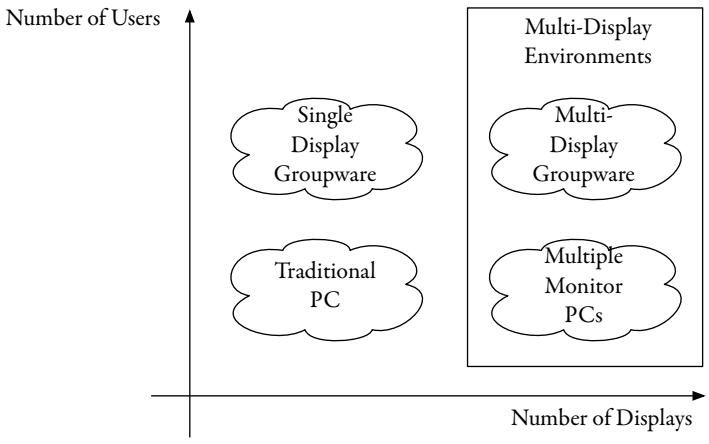

Figure 2.1: Dimensions in Display Environments

2.1 Single User, Multiple Displays

The most common multi-display scenario consists of a single user interacting with a single multi-headed computer, although a single user interacting with multiple computers is also possible.

2.1.1 Usage modes

There are two current implementations for multi-display use on a single PC: mirrored mode and extended desktop.

2.1.1.1 Mirror / clone mode

Replicating the same information across two or more displays is known as *mirror* (Mac OS) or *clone* (Windows PC) mode. This replication is primarily used in situations where two display devices differ physically, such as when a projector is plugged into a laptop, as illustrated in Figure 2.2. In this case, the physically smaller built-in laptop display may be used by the presenter to interact with the laptop, while the audience sees the screen contents mirrored on the much larger projected display that is often out of the presenter's field of view.

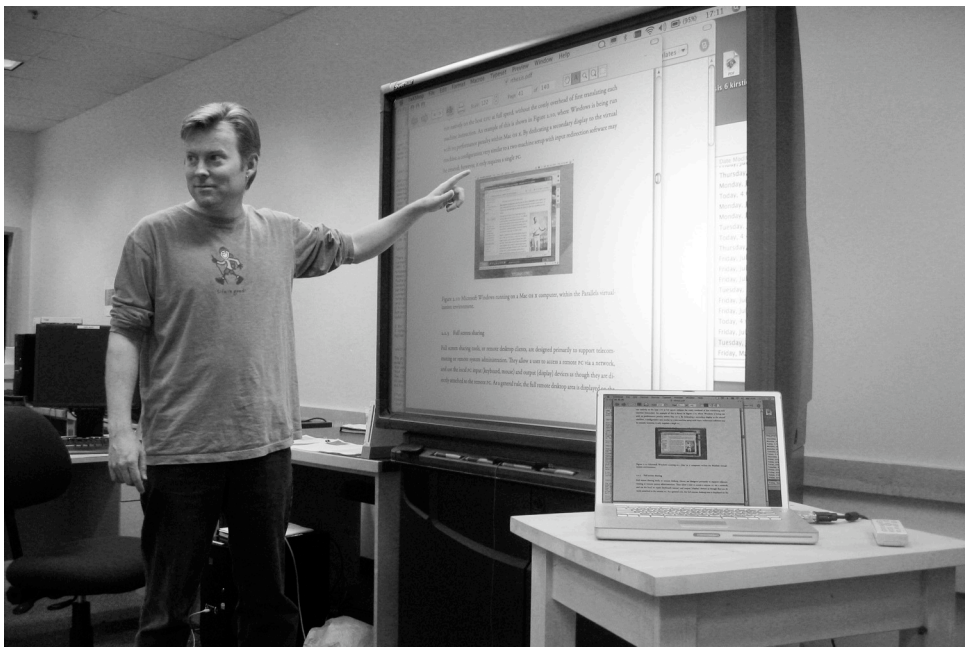

Figure 2.2: Laptop with external display operating in Mirror / Clone Mode. In this mode the image being externally displayed is an exact replica of what is seen on the built-in flat-panel display.

Mirror mode can also be achieved with a hardware video splitter, and indeed many modern projectors now include a built-in video splitter to allow desktop PCs without dual-head capability to be used in this manner.

2.1.1.2 Extended desktop mode

In the extended desktop multi-display configuration, each of the displays attached to a PC is represented by its own non-overlapping region in global screen space. Figure 2.3 shows two displays configured in extended desktop mode, with a GUI window spanning them. Both the Mac OS X and Windows operating systems restrict the position of each screen within global screen space so that the screens must touch along at least one edge; this allows the mouse cursor to travel seamlessly between regions of the extended desktop. In a configuration with three or more displays, some screens might touch more than one other screen, and this could occur along more than one edge.

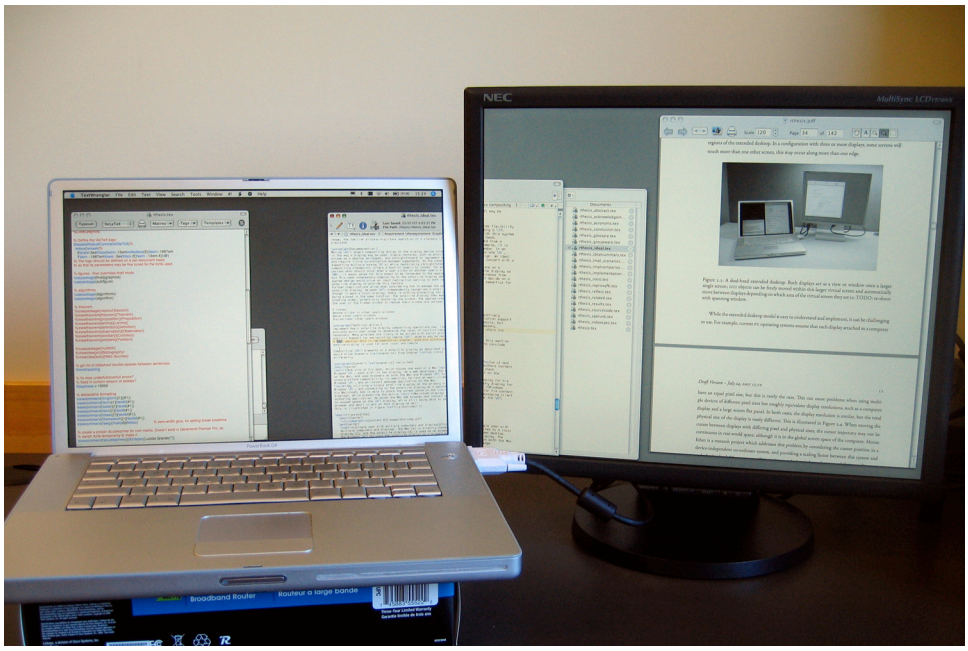

Figure 2.3: A dual-head extended desktop. Both displays act as a view or window onto a larger virtual screen space; GUI objects can be freely moved within this larger virtual screen and automatically move between displays depending on which area of the virtual screen they are in. In this example, one GUI window straddles the two displays, with different portions shown on each.

While the extended desktop model is easy to understand and implement, it can be challenging to use. For example, current PC operating systems assume that all displays attached to a computer

have an equal pixel size, but this is rarely the case. This can cause problems when using multiple devices of different pixel sizes but roughly equivalent display resolutions, such as a computer display and a large screen flat panel. In both cases, the display resolution is similar, but the total physical size of the display is vastly different. This is illustrated in Figure 2.4. When moving the cursor between displays with differing pixel and physical sizes, the cursor trajectory may not be continuous in real-world space, although it is contiguous in the global screen space of the computer.

Mouse Ether is a research project that addresses this problem by considering the cursor position in a device-independent co-ordinate system, and providing a scaling factor between this system and each display [11]. Mac OS 10.5 will introduce resolution independence capabilities, scaling not only the cursor position to a device-independent co-ordinate system, but the size of GUI elements as well [4].

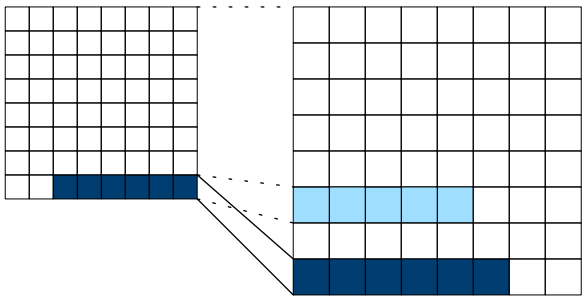

Figure 2.4: Pixel size differences in display devices. Both devices have an equivalent display resolution of 8 × 8 pixels. The *computer display* on the left has a real-world size of 1″ per side, while the *large screen display* on the right is 1.5″ per side. A dark horizontal line drawn by the computer has a vertical discrepancy where it crosses displays: in logical space, the output on the large screen display should be scaled down, so that in physical space the line would continue where the light pixels are shaded in.

Additionally, Mouse Ether allows the cursor to exist in the interstitial space between displays that have been arranged diagonally, which eases transitions by enlarging the portion of a screen edge that may be used to convey the cursor. Figure 2.5 illustrates this concept. Perspective Cursor is another research project which uses a 3D model of the planar displays in the environment coupled with a head tracker to dynamically compute the appropriate relationship between mouse and cursor movement across all displays [54]. This system accounts for display angle perspective distortion relative to the user's viewpoint in addition to size differences.

A further difficulty with the extended desktop model arises when many displays are used at

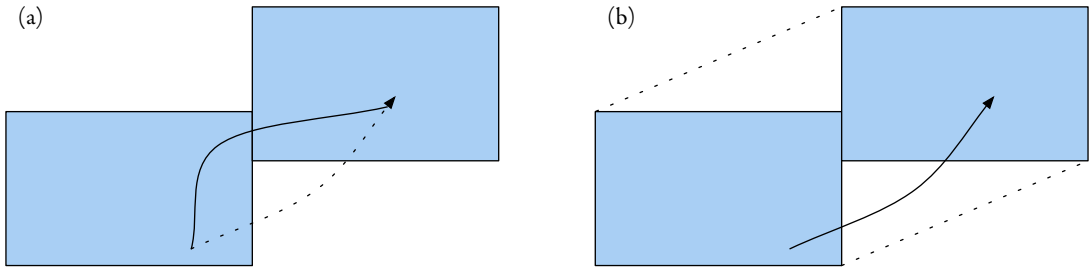

Figure 2.5: Mouse Ether. The traditional means of moving the cursor between two screens (a). The dotted line indicates the shortest path between two points, while the solid line indicates an actual path required to navigate the small portal joining the two screens. In this case, the cursor may only exist within the region defined by the screens. With Mouse Ether, the cursor may travel through interstitial space, indicated by the convex hull defined by the outline of the screens and the dotted lines (b). In this case, the user may directly follow the shortest path with the cursor, as indicated by the solid line.

once. As the size of the desktop grows, so does the distance between the GUI objects a user may want to interact with, which in turn increases their acquisition time [45]. By providing a warping function that jumps the cursor between displays, Multi-Monitor Mouse simulates having one cursor per display while retaining a single physical pointing device [12]. The warping function is invoked by pressing key or mouse-button combinations, by the location of the user's head, or by the location of the physical mouse in relation to the surface it is being used on. For an example of the latter mode, a system with three displays would have three mouse pads, one corresponding to each display. As the user moved the mouse between pads, the cursor would switch displays.

None of the research projects surveyed in this section have been released as products. As such, it is difficult to determine the overall impact these concepts may have on real-world multi-display usage. Because the satellite display developed in Chapters 4 and 5 builds orthogonally to these projects, it could benefit equally from them.

2.1.2 Multiple display hardware

The earliest IBM PC multi-display setups with different information on each display debuted in the early 1980s. This predates the introduction of Microsoft Windows by several years. These PCs often contained one each of a colour and monochrome graphics card. This dual card arrangement exploited the fact that each type of card mapped its framebuffer to a different region in memory, allowing simultaneous use of both types of graphics. While not technically a seamless extended

desktop—users couldn't drag a window from the colour display to the monochrome display—it did allow for related information to be displayed. Common configurations showed spreadsheets on the monochrome display with graphs and charts on the colour display, or debugging information on the monochrome display for the application being run on the colour display [26].

The first personal computer to support multiple displays in the now familiar extended desktop format was the Macintosh II, introduced in 1987 [81]. This was made possible by the NuBus expansion bus, which allowed multiple graphics cards to be installed in a single system.

Prior to Windows 98, using multiple displays with a Windows PC required specialty multi-headed graphics cards and drivers. Their high cost limited their use to a small market of business users such as engineering and financial institutions. With the introduction of Windows 98 and the PCI expansion bus, it was possible to install multiple graphics cards and Windows natively supported them. The extended desktop configuration was provided by Windows for additional PCI graphics cards, while clone mode and other settings were the responsibility of the card vendor, requiring separate utilities and drivers [67].

Due to performance restrictions of PCI, the PC graphics architecture switched to a dedicated graphics bus, the Accelerated Graphics Port (AGP), during the era of Windows 2000 and XP. To gain the necessary performance, early AGP specifications only supported a single *port*, or card, per system. This introduced an asymmetry between the primary graphics card and any secondary cards running in a slower PCI slot, which once again echoed the pre-Windows 98 systems in terms of multi-head capability. To compensate for this, graphics card manufacturers began producing AGP cards with two to four outputs to drive multiple displays [46]. As of 2006, an updated PCI bus (PCIe) has eclipsed AGP performance, and it is once again possible to run two or more graphics cards at the same speed. Current trends indicate that cards with multiple outputs will continue to be produced, thus making it easier than ever to run more than two displays from a single computer.

Although the new PCIe bus allows a larger number of displays to be connected to a desktop PC, it does little for laptops. The multi-display expansion of these small PCs continue to be constrained by other hardware factors, regardless of the bus technology used.

2.1.2.1 Current developments

In recent years, the popularity of extended desktop setups has created a market for devices and software that offer similar functionality to multiple graphics cards or multi-headed cards, while making use of existing surplus or under-utilized hardware.

In 2005, Matrox introduced a series of *Graphics eXpansion Modules*, devices that allow any computer with a single VGA output to drive two or three displays [47]. By electrically simulating a display with a resolution two or three times as wide as normal, the device appears as a single display to the computer. The expansion module subsequently splits the wide image across the multiple attached displays, as shown in Figure 2.6.

Figure 2.6: Matrox DualHead2Go. This device accepts a video signal at display resolutions twice as wide as a normal display, and logically splits the image for output to two display devices, each having normal width.

The Matrox GXM offers several advantages beyond simply allowing computers with a limited number of video outputs to support additional displays. Because the device interfaces at such a low level, none of the software running on the computer needs to know about the multi-display nature of the final output. Task- and menu-bars, traditionally restricted to a single screen of a multi-screen desktop, now span all displays. Windows applications that require full-screen hardware-accelerated graphics contexts such as 3D renderers, or media players with hardware video decompressors, can now extend content across multiple displays, instead of being limited to only one of the attached displays. The GXM is not without drawbacks, however. The device continues to utilize the analogue VGA graphics protocol at a time when the majority of display devices are now digital flat panels. In addition, the GXM can only split an extra wide screen horizontally, resulting

in side-by-side displays. While this is fine for many tasks such as graphics editing, being able to split a tall screen into two vertically stacked displays would be useful for text-centric tasks such as programming.

Just as dual-head graphics cards make effective use of old or otherwise surplus displays [29], commercial software products have recently started to appear that allow the use of a surplus network-attached computer—notably laptops with fixed displays that can't be plugged into a normal graphics card—in a similar manner [10, 87, 91]. One such product, MaxiVista, is shown in Figure 2.7. These products provide a virtual display on a user's computer, and a network display server on a display computer. When the software is running, the server takes over the full screen of a display computer, and shows the GUI objects that are moved to the virtual display by a client computer.

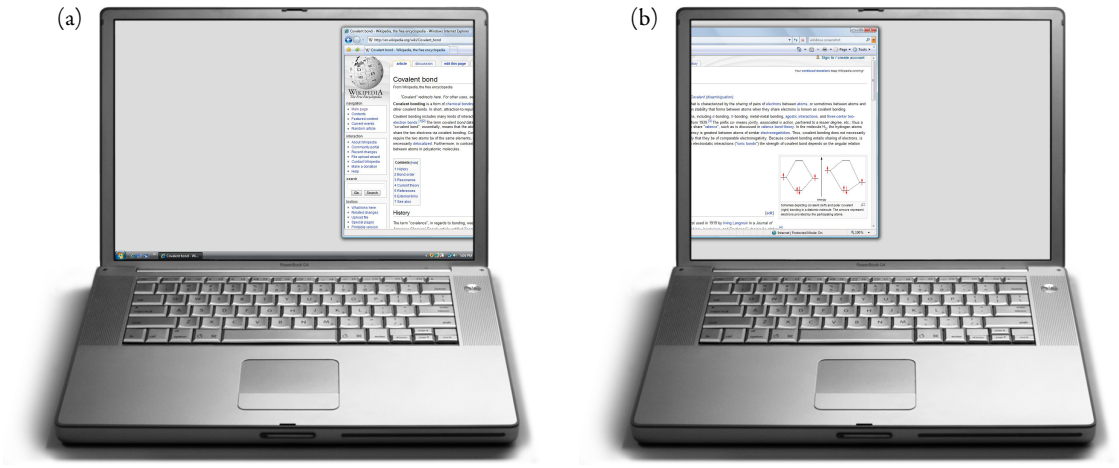

Figure 2.7: Maxivista. The user's computer (a) has a *secondary display* attached through the use of a special graphics card driver; anything that is rendered to the proxy secondary display is captured, sent over a network, and displayed on a different computer (b).

The commercial exploration of the virtual hardware on a *soft display* market has primarily been fueled by resource preservation: reusing existing hardware. While this is a laudable goal, little has been done to use the donor machines as anything more than display devices, which overlooks a substantial amount of computing power driving the displays.

Utilizing either wired or wireless networking as a transmission media provides additional flexibility over the traditional point-to-point connectivity of graphics devices provided by the DVI and VGA protocols. For example, multiple PCs could connect simultaneously to a soft display, which would then tile or composite the incoming graphics as appropriate.

It has been recently reported that Samsung will soon start offering a display that uses USB to receive graphics data from a computer, rather than a DVI or VGA connection [64]. The marketing information claims support for up to five displays per computer, depending on the application.

2.1.3 Multiple computers

Single users often control multiple computers: a web designer may use two different platforms to test the browser compliance of a project, or a software developer may build on one machine while debugging on another. This often leads to multiple sets of keyboards and mice cluttering a desk. A solution to this problem is input redirection, a mechanism to switch a single input source among the various computers.

By using only the keyboard and mouse switching facilities of a KVM and leaving displays connected directly to each computer, an extended desktop configuration may be simulated in hardware. This approach requires the use of a key combination or external button to transfer control between machines, but has the advantage that it requires no software to be installed on the machines, or any networking infrastructure. In the former case, the KVM sniffs key events sent by the keyboard and filters out appropriate hot-key commands before they are sent to the computer. These commands are then used to activate the KVM switching mechanism.

Software products such as `x2x` [17], `x2vnc` [35], Multiplicity [86], Teleport [76], Desktop Rover [58], and the cross platform `synergy` [84] allow a user to move a single cursor and input focus between computers by simply dragging the cursor as though all of the displays belonged to a single multi-headed machine. One machine is designated as a master, and has a mouse and keyboard plugged in to it. The rest of the computers are slaves, and don't require input devices to be directly connected. The software then captures input events on the master computer, and forwards them as appropriate to the slave computers over a network. The resulting simulation of an extended desktop isn't perfect; for example, only control is transferred between machines and moving GUI objects such as file icons or windows between displays is often not supported. Of the tools listed above, only Teleport allows the user to drag a file icon to initiate transferring the file to the remote machine.

Note that the hardware and software techniques are not mutually exclusive: combining both into a single system allows the ease of use of an extended desktop style interaction, while allowing a fallback to hardware KVM operation should the user need to interact with a secondary system while software or network resources are unavailable. This is illustrated in Figure 2.8, where `synergy` is

used once the systems have booted to the GUI , and a KVM is used to access boot-time options on both machines.

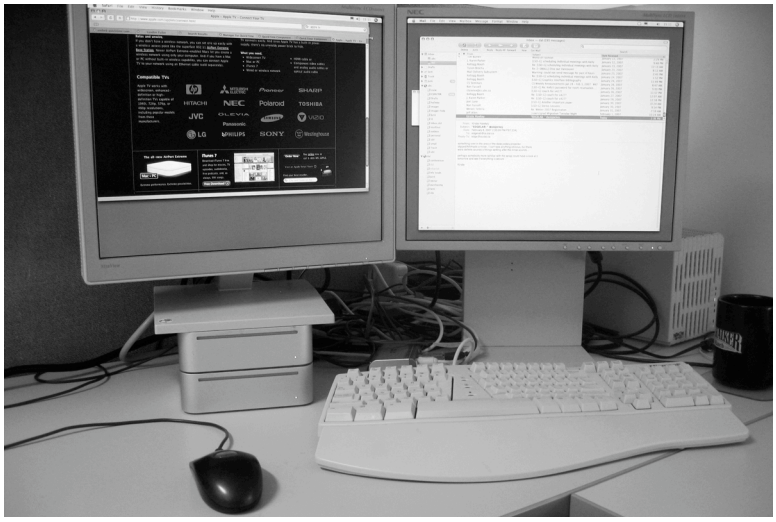

Figure 2.8: Multiple computers sharing one keyboard and mouse. In this figure, both hardware (a KVM) and software (the `synergy` input redirection tool) are used to transfer input events between computers, depending on the task.

2.2 Single User, Single Display

While the focus of this thesis is on MDEs, there are a number of relevant single display projects that aim to emulate multiple displays. Entries in this section consist primarily of virtual multiple display solutions, spanning both hardware and software. Falling into two broad categories, the first type switch or virtualize displays, providing serial access to multiple devices. Products in the second category are designed to access remote computers over a network.

2.2.1 Technical background

Figure 2.9 is a sequence diagram of the events that cause an image to be shown on a display. This diagram represents a simplified version of a compositing window server; a traditional window server operates in a similar manner, but has more synchronization between the different steps.

As an application runs, events may trigger changes in its visual state. These include external events such as the receipt of network data, or GUI-based changes such as scrolling within a window. Any event which may change the visual state of the application notifies the window server that a

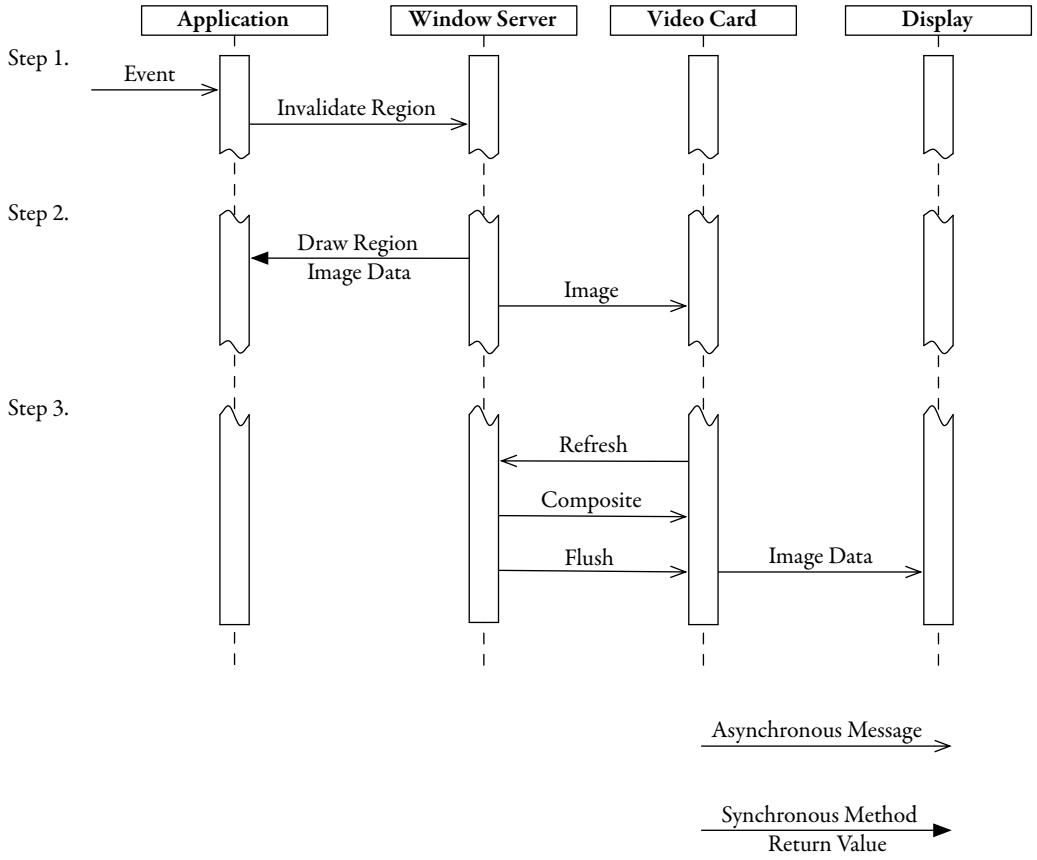

Figure 2.9: Sequence of events causing an image to be drawn to a display device.

region has become *dirty* or invalidated, and needs to be redrawn to the display (Figure 2.9, Step 1). The window server then asks the application to draw the changed image in a region of memory, in the case of a compositing window server this memory is an off-screen portion of Video RAM (VRAM)(Figure 2.9, Step 2). Finally, the graphics card contains a clock which is synchronized to the display hardware. During a *blanking* phase, when the display hardware is not drawing, the graphics card notifies the window server. The window server then composites the visible windows into the primary framebuffer, which the graphics card translates into an electrical signal to be sent to the display device. Step 3 in Figure 2.9 illustrates this procedure. If the application's visual state has not changed between successive refreshes of the display device, Steps 1 and 2 are skipped, and the window server uses the previously buffered image during the compositing phase.

Low-level graphics interface standards such as DVI and VGA also influence the ways displays are configured in an MDE. Such interfaces offer point-to-point connectivity, which has implications

for connecting large numbers of displays to a single PC. A technical summary of current graphic interface specifications is given in Appendix C.

2.2.2 Virtual displays

When used as intended, KVMs switch one display device across multiple graphics outputs. In the process, they turn a single display into a virtual set of multiple displays by removing the physical task of manually reconnecting the single display as it is needed on different computers. In addition to handling graphics data, KVMs also switch input devices, and some newer ones switch audio signals and other peripherals such as USB devices.

Virtual window managers such as ROOMS [34], swm [43], and FVWM [55] offer a software implementation of a simulated extended desktop. They can be thought of as providing a single window or viewport onto a series of desktops, each the same size as the primary display. By using a hotkey or mouse gesture, the current desktop may be switched for a hidden one, yielding serial access to the full extended desktop. The virtual window managers offer the task partitioning benefits of a multi-display system on a system with a single display. This saves the expense of both acquiring and housing additional physical hardware. There is typically a *pager*, or iconic representation of the full extended desktop, on each desktop. This GUI widget offers random access to each desktop, as well as providing a rapid means of moving windows and other GUI artifacts between desktops. Arranging windows by task across multiple real displays allows users to partition and more effectively manage the different kinds of information they use [28].

Virtual window managers emulate one aspect of a computer—the ability to have a number of displays attached. In a similar way, application software is available that virtualizes or emulates not only the display output, but an entire computer [97, 48, 21, 63]. Virtualization takes advantage of the fact that many PCs share the same CPU family (Intel x86), so to emulate a different machine, only the I/O facilities need be emulated. This allows the CPU in the virtualized environment to run natively on the host CPU at full speed, without the costly overhead of first translating each machine instruction. An example of this is shown in Figure 2.10, where Windows is being run with no performance penalty within Mac OS X. By dedicating a secondary display to the virtual machine, a configuration very similar to a two-machine setup with input redirection software may be created, but it only requires a single PC.

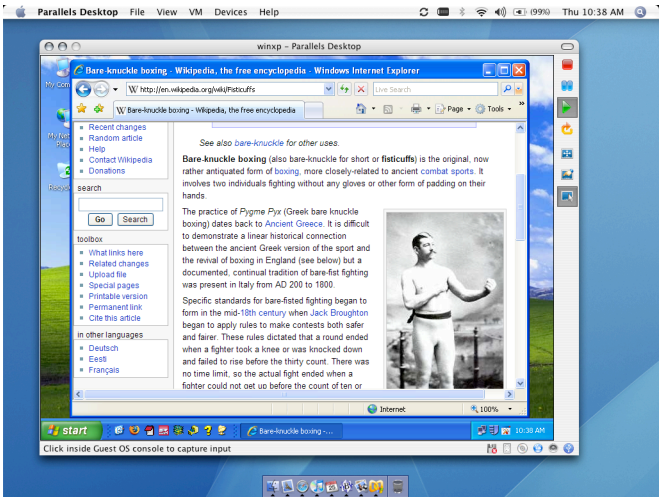

Figure 2.10: Virtualization Environments. Microsoft Windows running on a Mac OS X computer, within the Parallels virtualization environment.

2.2.3 Full screen sharing

Full screen sharing tools, or remote desktop clients, are designed primarily to support telecommuting or remote system administration. They allow a user to access a remote PC via a network, and use the local PC input (keyboard, mouse) and output (display) devices as though they were directly attached to the remote PC. As a general rule, the full remote desktop area is displayed on the local computer, scaled or panned as necessary if it is larger than the local display. Examples of this technology include VNC [70][75], Timbuktu [59], Microsoft Remote Desktop Connection and rdesktop [49, 18], and Apple Remote Desktop [2].

VNC is the most popular of the remote desktop tools, largely due to its lightweight and openly published remote framebuffer (RFB) protocol [74], which has been implemented on a large number of platforms. This gives it substantial cross-platform capabilities, in some sense simulating the ability to run a foreign OS locally, much as virtualization software does. UltraVNC [98] is a VNC server for Windows XP notable for a single-window sharing feature: unlike other full screen sharing tools, it can bound its view of the remote display with the dimensions of a single window, tracking that window around the remote screen if necessary. This single-window support may reduce the bandwidth required by the stream, as well as use less screen real-estate on the target computer.

Single-window mode does not accurately simulate a seamless remote window, however. As the same framebuffer scrape method is used as a standard VNC server, overlapping window contents

may be captured as well. This problem is illustrated in Figure 2.11. Additionally, UltraVNC is limited to sending only one single window in this mode. This is due to the fact that the RFB protocol is designed to send a full screen–UltraVNC is simply configuring the underlying RFB transport to use a small full-screen size. The RFB protocol does not support the multiple independent video streams necessary to transmit multiple single windows simultaneously.

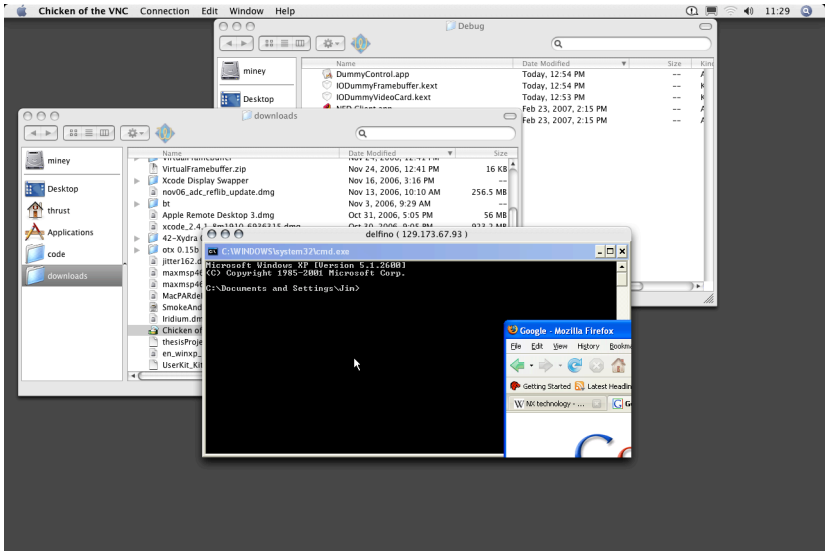

Figure 2.11: Single window sharing with UltraVNC. Note that the *single window* includes a portion of an Internet Explorer window, due to the way the Windows XP window server composites directly to the framebuffer.

In addition to the special purpose full screen sharing tools listed above, X Windows natively supports remote desktops and windows due to the client/server model inherent in its display server design [83].

2.3 Multiple Users, Multiple Displays

There are several co-located MDG research projects that informed the development of the multi-user concepts reported in this thesis.

ARIS is a window manager for a multi-display interactive space developed at the University of Illinois by Biehl and Bailey [14]. To move windows between displays in the space, a user selects a window and activates an attached iconic map, or *world in miniature*, of the space. The user then drags an iconic representation of the selected window within the map to an iconic representation

of the desired destination display within the map, at which point the real window is transferred between displays and input is redirected to the new display. This window movement is accomplished by migrating the running application between the PCs attached to the displays. Because ARIS runs on top of the GAIA middleware framework [79], which handles the application migration, applications must be re-compiled to run in the multi-display space provided by ARIS.

A different conceptual approach to moving objects between displays is used in iRoom, which runs on the iROS middleware framework developed at Stanford [66]. Rather than moving running applications between systems, data and state is stored in a central DataHeap. By sending events through an EventHeap, custom applications are started and stopped on the different PCs backing each display. For example, sending a `gotourl` message to a display automatically starts the correct browser for that display, and renders the webpage. Only objects of supported types can be moved using this mechanism.

The Dynamo system developed by Izadi *et al.* at the University of Nottingham is designed to provide *ad hoc* opportunistic collaboration between a community of users [40]. It accomplishes this by providing a large tiled display surface from which users may 'carve' non-overlapping regions. These regions may then be used for collaborative tasks, or they may exist beyond the active time of the user, enabling users to leave notes or files for other community members.

2.3.1 Multi-user input redirection

As with single user input redirection, there are many tools supporting multi-user input redirection. These include: MightyMouse (later renamed Rover due to possible copyright concerns) developed by Booth *et al.* at UBC [15]; PointRight developed by Johanson *et al.* at Stanford [41]; Pebbles from Myers *et al.* at CMU [53]; and Swordfish by Inkpen *et al.* at Dalhousie [30]. Key differences among them are the methods by which relationships between participating computers are specified and the methods by which switching is accomplished.

Pointright and Swordfish use a spatial layout and mouse traversal to move between remote screens. Figure 2.12 illustrates the MightyMouse (Rover) toolbar, which is used to warp or jump the cursor to a remote display. The cursor is always warped back to the user's local screen before moving to another remote display: this allows the user to traverse a large number of displays quickly, without interrupting other users. The Pebbles project differs from the others by allowing the use of PDAs rather than PCs to control a remote computer.

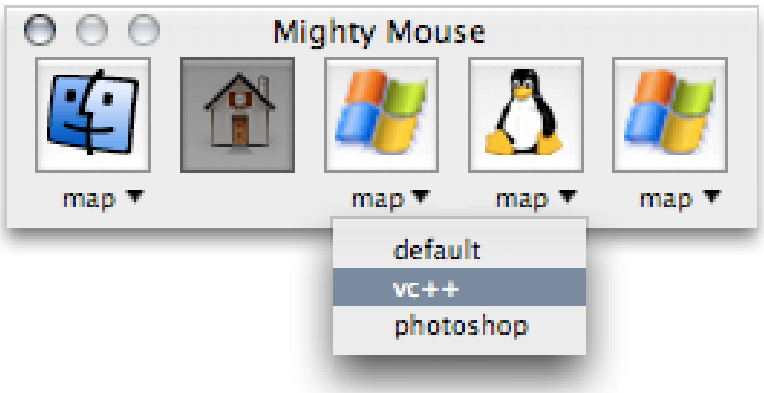

Figure 2.12: The Mighty Mouse (Rover) toolbar. This toolbar is used to move the cursor between PCs in the system. The dropdown menu shows the cross-platform keyboard translation mapping in use on the machine being controlled.

2.4 Other Technologies

There are other technologies that influenced the development of the tools reported in this thesis. They will be discussed here.

2.4.1 WinCuts

Wincuts is a tool developed by Tan *et al.* at Microsoft Research that allows the allows a user to select regions of a display for duplication to either the local screen or a remote machine [90]. By cutting a region of a window and displaying it on a remote screen, a remote extended desktop may be roughly simulated. Remote Wincuts are explicitly managed by a Visitor tool, which does not offer drag and drop placement between displays. That is, Wincuts does not emulate an extended desktop configuration, rather a pop-up menu on the Wincut allows it to be sent to a remote display where it may be manipulated by the Visitor tool.

2.4.2 X11

Introduced in 1984, X11 is the primary GUI windowing system for Unix based computers [83]. X11 is based on an internal client/server model, where the process that renders a GUI object to a screen (the display server) is not the same as the one that created it (the client). This stems from a

networked thin client architecture in which the computer on a user's desk is simply an I/O device for a central compute server.

Even though multiple thin clients may have been connected to a single server, there was no way to migrate the GUI artifacts of a running process between display servers. The X11 model provides multi-head displays in the form of multiple clients, but does not support extended desktop functionality. This capability is now provided by several extensions to X11. Xinerama is an extension to X11 that allows a single display server to work with multiple displays, providing an extended desktop to X11 servers that utilize multi-headed graphics cards [42]. Distributed multihead X (DMX) is a recent extension to X11 that allows multiple computers, each running a regular X11 server, to behave as though they are local displays for a primary PC [27]. These DMX displays can then be used with Xinerama to present a single large desktop (spanning multiple computers) to the user. One key limitation of X11 is the lack of dynamic configurability of the X server. To add or remove secondary displays, the X server must be restarted—a less than ideal situation for laptop users who frequently connect and disconnect from external displays.

2.4.3 Window servers

Window managers are responsible for creating the look and feel of a desktop environment, as well as providing GUI artifacts such as window borders, buttons, and menus. Window servers then handle the layout and compositing of these GUI objects into the framebuffer. In Microsoft Windows and Mac OS X, the window manager and window server are combined to form an integral part of the graphics infrastructure. In contrast, the window manager used by X11 is provided as a separate component that may be replaced with one of many different managers offering different looks and behaviours.

In the past, memory has been an expensive computer component. This is especially true of VRAM, which is dual ported to allow the system to write to the framebuffer while the graphics hardware simultaneously reads from it. Because of this, graphics cards specified a maximum depth and resolution they would support, and only included enough VRAM to hold a framebuffer of this size. As a result, operating systems (including Mac OS 9 and earlier, Windows up to and including XP, and most current X11 implementations) use a framebuffer window server to make the most of a limited resource. These window servers ask applications to draw directly to the framebuffer whenever window contents change, or when a portion of a window is uncovered by an overlapping window. This configuration trades memory expense for drawing expense, requiring content

to often be recomputed even if it has not changed.

With the advent of 3D acceleration on modern graphics cards, additional VRAM had to be included to store geometry and texture data. With graphics hardware now containing buffer memories much larger than what is necessary to hold a framebuffer, a considerable amount of memory was made available when the 3D facilities were not in use. Operating system developers then started to display windows with a two step process: first by rendering windows to offscreen VRAM, followed by using graphics hardware to composite the offscreen windows to the framebuffer. Decoupling the application rendering from final drawing has several advantages. For example, applications do not need to re-draw static content whenever their windows are invalidated (uncovered by another window, or reshown after being entirely hidden). In addition, visual effects may be applied by the graphics hardware during the compositing step, including alpha blending and blurring. Operating systems that support compositing window servers include Mac OS X [62], Windows Vista [82], and X11 with the Metisse, Metacity and Compiz [19, 65, 73] window managers.

2.4.4 Multi-pointer X

The Multi-pointer X (MPX) project by Hutterer and Thomas at the University of South Australia provides true multi-cursor support to both legacy (single user) and SDG applications [39]. To accomplish this, the X11 server was modified to create individual core cursors for each mouse device, rather than multiplexing the data from multiple devices into one cursor. The modified core cursors have an additional ID value that indicates which device is in use, the IDs can then be used to implement access control to specific GUI widgets in an SDG application. The MPX project makes no attempt to bind additional keyboards together in *input channels* [89], which would provide multiple insertion points for text entry.

2.5 Summary

As explained in Section 2.1.1, there are currently two usage modes for multiple displays connected to a single computer: mirrored mode and extended desktop. Section 1.1.1 in Chapter 1 indicates that there are an additional three modes available, if all possible 2D screen arrangements are considered. Chapter 3 presents a configuration utility that enables the additional three modes on a Mac OS X PC.

Using multiple displays with a single PC continues to be constrained, especially from a laptop.

The Matrox GXM and MaxiVista products described in Section 2.1.2.1 work to remedy this, but each of these solutions has its own limitations. As MDEs become more prevalent, choosing and using the most appropriate display for a task becomes more difficult. MDG suites such as ARIS, iRoom, and Dynamo were developed to address this, but they are heavyweight middleware operating systems that don't facilitate *ad hoc* usage. A key observation underlying our work is that display devices are naturally shared and foster collaboration, due to the ease in which multiple people can view a single device simultaneously. This simultaneous access does not extend to the content that may be displayed, however, because devices only accept input from a single source at a time. The satellite display project introduced in Chapters 4 and 5 aims to address these issues.

Chapter 3

Reflect: An Advanced Display Configuration Utility

As discussed in Chapters 1 and 2, the multi-display configuration capabilities made available in current operating systems are limited to the mirror and extended desktop modes. This chapter considers the technical underpinnings of a multi-headed graphics card, and argues that the availability of only two usage modes is the result of a design choice rather than a hardware limitation. The results of generalizing multi-display configurability beyond these two basic modes are presented in Section 3.3. Section 3.4 describes the implementation of a utility that allows for generalized multi-display configurability, and usage of the additional configuration modes is considered in Section 3.5. Finally, in light of the additional capabilities afforded by the implementation, the choice made by the original system designers to restrict the number of modes is reconsidered.

3.1 Background

Traditionally, multi-display configuration capabilities on personal computers have been restricted to **extended desktop** and **mirror mode**, or a combination of the two. Figure 3.1 shows a schematic representation of these two modes in screen shots of the Mac OS X display configuration utility. On the left, two screens are arranged in extended desktop mode, where they abut along a shared edge. The right hand arrangement indicates that the screens are mirrored, with a stacked representation. Figure 2.3 in Chapter 2 demonstrates the extended desktop mode in use with a single computer driving two displays, each displaying independent content.

Logically, displays are represented by axis-aligned rectangular regions called **screens**, on an abstract two-dimensional plane referred to as **screen space**. Screens are ordered, with the first screen having some distinction as the main screen while others have co-equal status as secondary screens. The main screen typically contains some global state management tools such as a menu bar and Dock (Mac OS X) or a task bar and Start menu (Microsoft Windows); the top left corner of this screen defines the position of the origin for the coordinate system used in screen space.

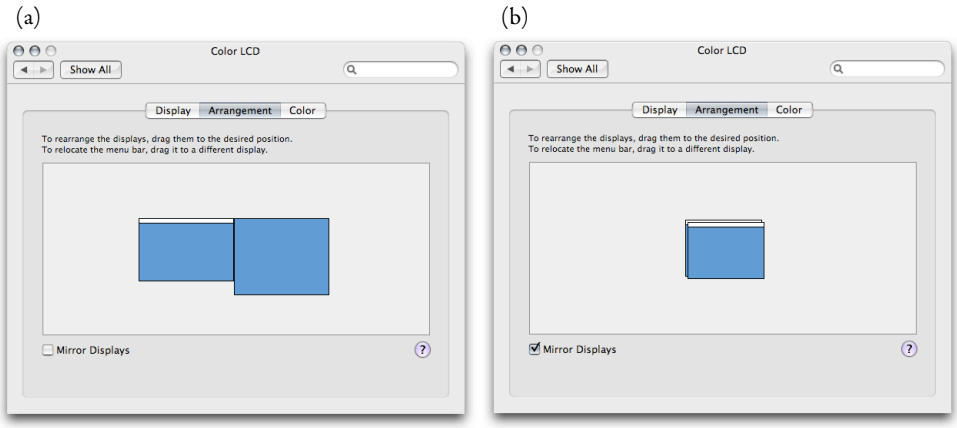

Figure 3.1: Configuring multiple displays on Mac OS X. Iconic representations of extended desktop (a) and mirror mode (b) are shown.

3.2 Hypothesis

Using a PC in extended desktop mode does not require the attached displays to be the same logical size, or to be arranged in strict rows or columns within screen space. This implies that the graphics hardware utilizes an independent framebuffer for each screen, rather than a single large framebuffer that encompasses all screens. Knowing this, it was hypothesized that the ability to configure displays more generally was being constrained by an OS software design choice, rather than a limitation of the graphics hardware itself. This project presented in this chapter was implemented on an exploratory basis to test this hypothesis. The results are what was directly experienced with this implementation, generalizations to other systems will be made where possible.

3.3 Generalization of Multi-Display Configuration

A more general configuration capability would feature arbitrary extended desktop positioning, including partially overlapping and non-adjacent screen placement, as illustrated in Table 3.1. Additionally, having two fully overlapped screens of differing resolutions (i.e., one screen inset into another) is a generalization of mirrored mode. All five of these possible screen arrangements continue to be limited by the fact that the underlying surface on which the screens reside is planar, and this fact remains for the visual aspect of the arrangement. However, in the mouse control space the screen surface may be wrapped around in both the X and Y directions to simulate a torus, which increases the generality somewhat further [92].

Table 3.1: Multi-display configuration modes

Mode	Gap Between Screens
Mirror/Clone[a]	Negative
Extended Desktop	None
Disjoint	Positive
Overlap	Negative
Inset[b]	Negative

[a] Mirror/Clone is a special case of Inset
[b] Inset is a special case of Overlap

3.3.1 Implications for cursor movement and image display

Changing the gap between two screens has implications for both cursor movement and image display. By creating a gap larger than a threshold distance based on mouse acceleration properties, a screen can be made unreachable via normal mouse movement because the cursor is not allowed to traverse the interstitial space between screens. Using an alternate method to warp the cursor between screens, such as the Multi-Monitor Mouse [12], allows the user to benefit from a larger workspace without the cursor accidentally transferring between displays.

A positive gap between two screens introduces logical space that may be used to compensate for the physical space that bezels impose between two monitors. An image spanning the gap will have a logical strip removed corresponding to the physical space between the active display areas, resulting in correct measurements between points on both displays, as shown in Figure 3.2. This result is similar to the OneSpace project by Robertson *et al.* [77], but operates system-wide on any GUI element, rather than simply on image data as is the case with OneSpace.

A negative gap has no effect on cursor movement between screens, but allows screens to be logically overlapped. Visually, elements (including the cursor) that exist in overlapping screen regions will be shown simultaneously on each display corresponding to the overlapped screens.

3.4 Details of the Implementation

Reflect is a utility that was developed for this thesis to explore the possibilities afforded by increased multi-display configuration options. This prototype software was written for Mac OS X 10.4 running on a PowerPC processor, and as such, much of the low-level discussion will focus on

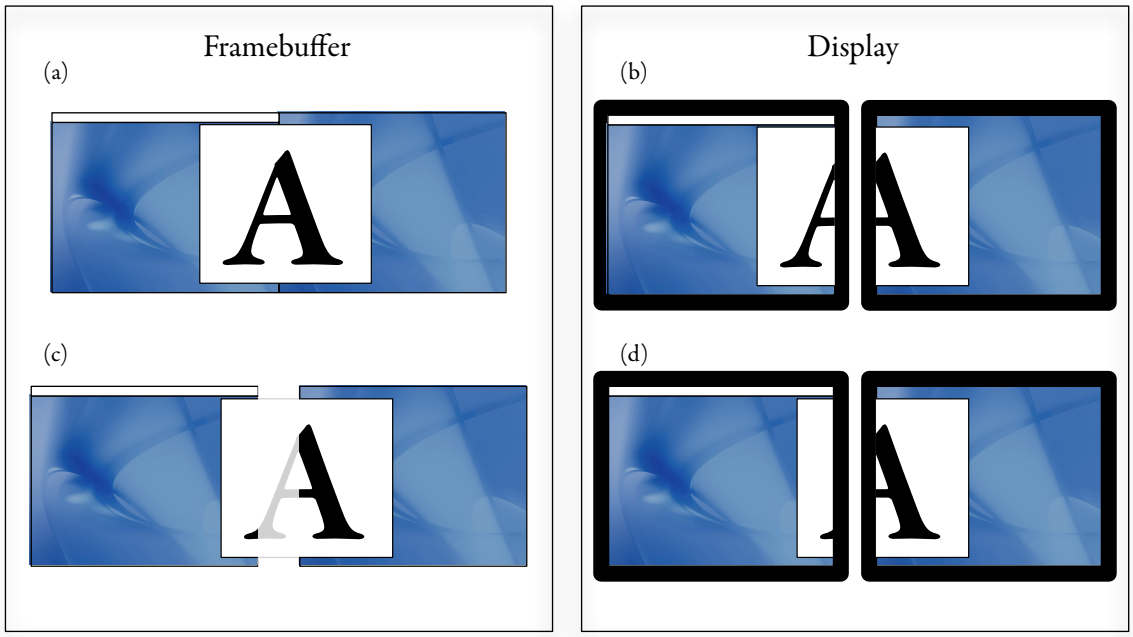

Figure 3.2: Effect of a positive screen gap on image display. For the upper image, a standard extended desktop is used that has two abutting screens in the framebuffer (a), and the physical dimensions of the image have been stretched when viewed on the two displays (b). By logically separating screens in the framebuffer (c), the bezels between displays can be compensated for, maintaining the aspect ratio of GUI objects spanning the displays (d).

this platform. Because the relevant underlying graphics hardware components are commonly used on multiple platforms, the extension of this project to other operating systems should be straightforward using similar techniques. Complete source code for the implementation developed for this thesis is available online; further details are available in Appendix D.

3.4.1 Technical background

Display devices are represented to an operating system as a framebuffer, or a contiguous region in VRAM. Multi-headed graphics cards have a single large region of VRAM which may be subdivided to provide framebuffers for all attached displays. In extended desktop mode, multiple non-overlapping **apertures** into the VRAM are defined, one for each display. Mirroring two or more displays is enabled by explicit configuration of the graphics card to use one aperture as the source for multiple display connections.

Creating arbitrarily overlapping screens is not simply a matter of overlapping the apertures,

as apertures and framebuffers are one-dimensional and require additional structural information to represent two-dimensional screens. To achieve the desired two-dimensional overlap, it must be enabled higher in the graphics pipeline, at the final level before the window server draws to the framebuffer. As the window server draws the contents of each screen to its respective framebuffer, the overlapping regions must be drawn multiple times, once per screen. If two overlapping framebuffers exist on the same graphics card, this duplicated drawing is a less expensive operation than one might initially think. windows are first drawn by the window server into offscreen portions of VRAM, the subsequent copy or compositing operation is performed entirely on the graphics card using hardware accelerated routines. If multiple graphics cards are used to add multiple displays to a PC, the window server may draw to each card. This configuration may not be accelerated as much as if both displays were attached to the same graphics card, however.

3.4.2 Approach

Apple exposes a public Application Programming Interface (API) in Mac OS X, called Quartz Display Services, to allow 3rd party developers to configure and control displays, including multiple display layout. This API is part of the CoreGraphics framework, which is used to manipulate graphics on Mac OS X. It was hypothesized that these calls are also used internally by the built-in display configuration utility. To allow arbitrary extended desktop screen placement, the following approach was used. First, the API calls in Quartz Display Services were tested to determine if they would allow arbitrary desktop screen placement. It was found that the API calls restrict screen placement in multi-display mode. To work around this limitation, the API calls were patched to allow arbitrary desktop screen placement. Once the API calls were adjusted, the built-in configuration utility was tested to determine if it would work with the patched API. It was found that the built-in configuration tool did use the patched API, and that it also contained its own code to constrain screen placement. Rather than further patching the configuration tool, a replacement configuration utility was written to take advantage of the unconstrained screen placement capabilities of the patched Quartz Display Services. These steps will be described more fully in the sections that follow.

3.4.3 Examining the public Quartz Display Services API

As a first step in determining the capabilities of the Quartz Display Services, a small test harness was written for the function `CGConfigureDisplayOrigin()`. The function `CGConfigureDisplayOrigin()` is used in Mac OS X to move a screen around in screen space, given new coordinates for the origin, or top left corner, of the screen. The test application verified that calling the function `CGConfigureDisplayOrigin()` successfully moved the screens, but did not allow them to be placed at arbitrary positions. Instead, the function `CGConfigureDisplayOrigin()` includes a check that changes the requested origin co-ordinates to the nearest valid position before moving the screen. In this case, a position is valid if there is no gap between screens.

3.4.4 Locating and patching `CGConfigureDisplayOrigin()`

All public API calls to manipulate displays are safety-checked internally [61]. In the case of the function `CGConfigureDisplayOrigin()`, the private function `CGXLFindBestPositionForDisplay()` determines if the requested screen position is valid, and if not it adjusts the requested position to the nearest valid position. To allow calls to the function `CGConfigureDisplayOrigin()` to succeed regardless of whether the requested screen position is valid, `CGXLFindBestPositionForDisplay()` needed to be patched to return success without executing its own code to move the screen to a valid location. A tool called `patchcg` was developed for this purpose. See Appendix B for an explanation of how the private safety-check functions were discovered, and how `patchcg` was developed.

3.4.5 The Reflect configuration utility

Once the CoreGraphics framework had been successfully patched to allow arbitrary screen placement, the built-in configuration utility, `Displays.prefpane`, was re-tested and found to also constrain screen placement. In addition, without control over the z-order of the iconic screens in `Displays.prefpane`, managing overlapping screens would be problematic. Instead of further patching, a proxy configuration utility called Reflect was created to allow the user to manipulate displays in a familiar manner, while affording them the new capabilities enabled by the CoreGraphics patch. Figure 3.3 shows Reflect in use on a three headed system, with both inset and disjoint screens being deployed.

When a new configuration is chosen using either Reflect or the system configuration utility,

the OS blanks all displays temporarily as the new layout is set. Because of this, the display configuration cannot be set dynamically. For example, an external display cannot follow or remain centred on the mouse cursor.

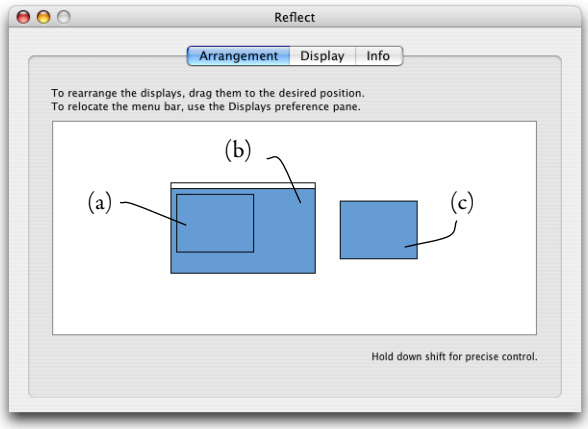

Figure 3.3: The Reflect configuration window. This view behaves similarly to the built-in Display Arrangement preference panel, but allows display proxies to be dragged anywhere in screen space. In this example, three displays have been configured, one properly inset (a) in the main display (b), and one disjoint off to the right (c).

3.4.6 Testing

Ad hoc testing indicated that many applications worked as expected within overlapped screen regions. Due to the depth at which the operating system was patched, and the way it was hypothesized that overlapping regions were drawn by the operating system, end-user applications should have no awareness that the configuration is non-standard. A brief survey revealed that applications such as Microsoft Excel and Adobe Photoshop, which may use custom widget toolkits or drawing routines, displayed minor rendering artifacts, perhaps as a result of trying to draw directly to the framebuffer instead of using high-level drawing APIs. Notably, multimedia applications such as 3D games and DVD players, which use extended features of the graphics hardware, often worked fine in overlap mode.

3.5 Usage Scenarios

The flexibility with which screens can now be configured leads to several previously unachievable usage scenarios as outlined below. Through these mechanisms, Reflect provides new solutions for Scenario 1 that was outlined in Chapter 1.

3.5.1 Inset mirroring

Mac OS X offers a mirrored mode in addition to the extended desktop configuration. In this setup, two (or more) displays will show exactly the same image, as though a hardware video splitter had been inserted between the computer and displays. This mode is held over from the time before projectors with built-in splitters were available, where it was useful to drive a data projector and a console display with the same information.

In the default mirroring mode available on Mac OS X, both displays are limited by the largest common size shared between the two. For example, if a projector can handle a maximum resolution of 1024×768 pixels, and a built-in laptop display has 1280×854 pixels, when mirroring these devices, the mirrored set has a resolution of 1024×768 pixels and the output must be stretched (to 1280×854) or inset on the built-in laptop display.

By using Reflect to configure multiple displays, one display may be used to mirror just a subset of another, rather than both displays being constrained to share a common resolution. This feature could be used in a lecture or meeting situation, where a portion of the main display is mirrored to a projector for demonstration purposes, while peripheral information such as notes are not. This is shown in Figure 3.4. In this example, an LCD panel is being used to simulate a projected display. A programming lesson is being developed in window 1a, which is mirrored on the external 'projected' display as window 1b. Windows 2 and 3 aren't shown on the external display. They contain a previously written working implementation of the code (window 2), as well as some notes (window 3). By limiting the scope of the mirroring, the audience can focus on important details, while distractions such as instant message or email notifications may be hidden in the extra space around the mirrored region. This allows the presenter to maintain some level of privacy [13, 90].

Graphic artists may use inset mirroring to provide a digital magnifying glass, in which a secondary display with a lower resolution and larger physical size can be used to inspect pixel-level details of a subregion of the primary display. Figure 3.5 illustrates this scenario, where an external

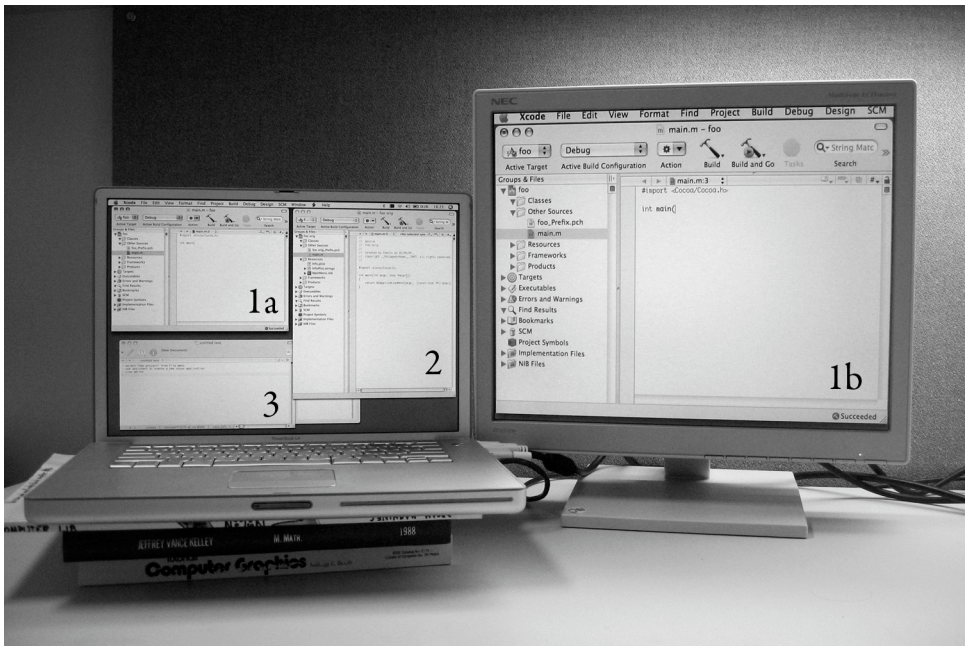

Figure 3.4: Inset mirroring example. The external display (1b) only shows the top left corner (1a) of the laptop's built-in display. This allows a portion of the screen to be mirrored, while omitting unnecessary clutter.

display is being used to show a magnified version of an area on the laptop display.

3.5.2 Overlapping displays

The overlapping capabilities of Reflect allow for an arbitrary portion of two screens to be overlapped, from extended desktop through to mirrored (or inset). This may be useful for projected displays arranged in a matrix, where the image produced by two projectors can also be overlapped. With additional alpha blending around each screen, two projected images may be *feathered* together to increase the seamlessness of the final image [68].

An additional use of overlapped screens may be considered in light of the *lost cursor problem*, where a cursor may accidentally switch displays when the user is attempting to interact with a GUI widget near the boundary between screens. For example, in a two display configuration there may be a scrollbar at the right hand edge of the left display. The user may overshoot the scrollbar with the cursor when navigating to it from the left, effectively losing the cursor on the right hand display. Now the user must move the cursor back to the left hand display, and attempt to reacquire the scrollbar again from the left. By overlapping the screens a small amount, the edge of each screen

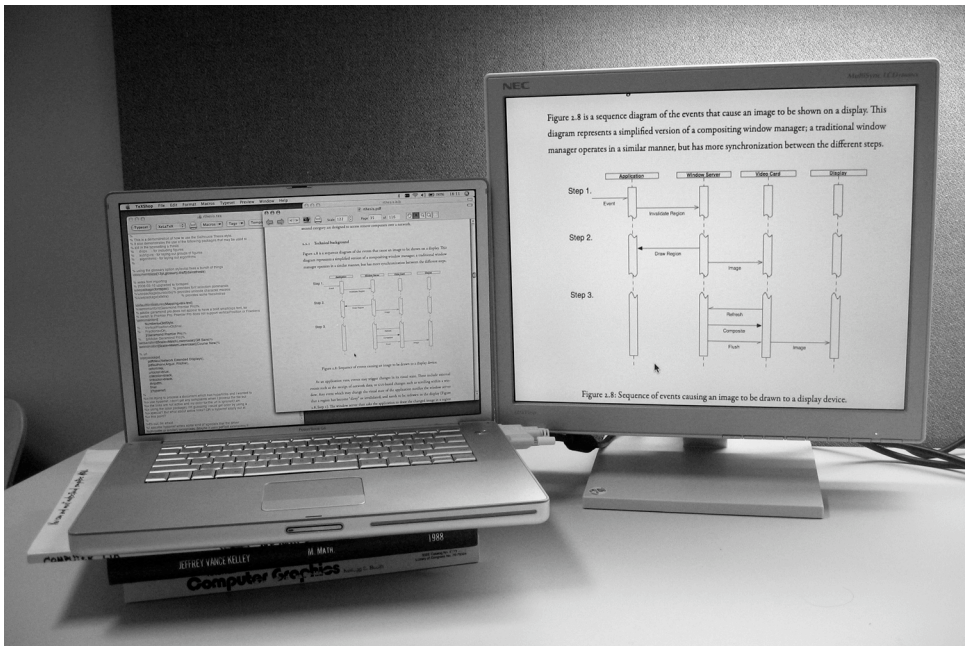

Figure 3.5: Inset mirroring used as a magnifier. The external display shows a zoomed in subregion of the laptop's built-in display.

is in effect shared by both displays. This is shown in Figure 3.6. Now the scrollbar exists on both displays, allowing it to be acquired from both the left on the left hand display, or from the right on the right hand display.

3.5.3 Disjoint displays

Separating two screens with a positive gap has the greatest effect on user interaction, as this alters the mouse cursor control space as well as the display space.

3.5.3.1 Display edge sticking

Rodgers et al. showed that introducing a slight *stickiness* as the cursor transitioned between displays in an extended desktop configuration improved user performance [78]. This behaviour is provided in Reflect as a side-effect of the augmentation to the desktop model. Allowing the user to position screens such that there is a slight gap between them interacts with the algorithm used to update the cursor position by allowing the user to position screens such that there is a slight gap between them.

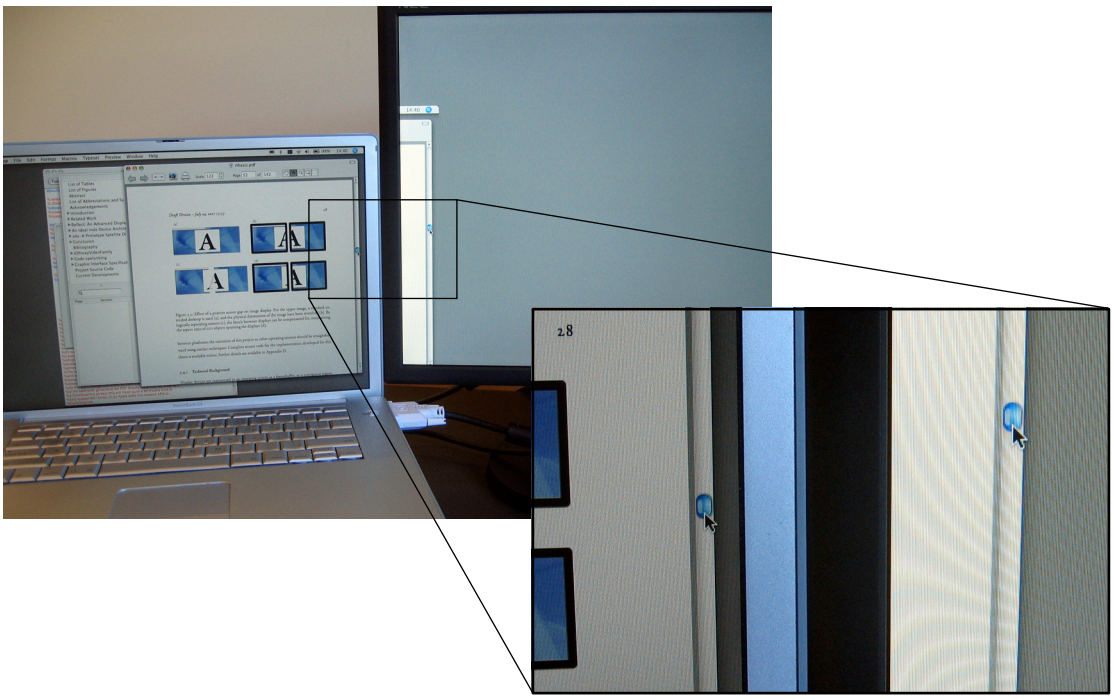

Figure 3.6: Extended desktop with partial overlap. In this example, the scrollbar exists on both displays, allowing the user to easily acquire it from either the left or the right.

The computation currently utilized by Mac OS X to move the cursor, on a single display or between displays, is implemented in the function `IOHIDSystem::_setCursorPosition()` in the IOHIDFamily drivers [3]. Each time the mouse is moved, this function computes the new location of the mouse cursor within global screen space, and searches the attached screens to determine if the cursor is within at least one of them. If so, the cursor is moved to the new location. However, if the cursor has moved outside the set of all screens, its location is clipped or scissored to the screen it originated from.

With a gap between screens, a mouse delta value must be larger than the gap to cause the cursor to move to the next screen. Small deltas simply pin the cursor at the previous position, simulating the behaviour of an exterior edge. It is not possible for the cursor to exist within the gap, it must either remain on the screen it was on before a cursor moved event, or transition to a new screen after the event. On current systems, displays with sizes between 1000–2000 pixels per axis are typical. Coupled with a cursor refresh rate of 60Hz, mouse deltas on the order of 20 pixels per axis are not uncommon during fast mouse movement to achieve a reasonable relationship between hand movement and the corresponding cursor motion. Thus with a gap between screens of

approximately 15 pixels, slow and precise mousing near the display edge does not generate deltas large enough to span the gap and the cursor is bound to the current display, but moving the mouse quickly from display to display easily overcomes the gap threshold and the cursor transfers from one display to the next as expected. The advantages of this approach are illustrated by re-visiting Scenario 1 from Chapter 1.

Scenario 1 revisited, changes in bold

> Alice uses a laptop as her primary computer, although it is most often used on a desk both at work and at home. In these locations, it is augmented by peripheral devices such as a keyboard, mouse, and external display. Because it is larger, Alice uses the external display as the primary screen in these situations and uses the built-in LCD panel as a secondary workspace to display notifications such as new email alerts. Alice finds the extended desktop configuration troublesome, as she often overshoots scroll-bars or other GUI widgets with her cursor on the primary external display, losing the cursor onto the laptop's built-in LCD. **By using the Reflect utility, Alice solves this problem. She now connects an external display to her laptop for use in extended desktop mode. As usual, she wishes to have the external display to the left of her built-in laptop display. Rather than simply using the display configuration utility provided by the OS to abut the two screens, she uses Reflect to create a 15 pixel gap between them. Now when she navigates to a scrollbar at the right-hand side of the external display, the gap between screens prevents her cursor from accidentally moving to the laptop display.**

3.5.3.2 Unreachable displays

Extending the display edge stickiness concept further, screens can be placed in screen space such that the gap between them is greater than the largest possible mouse delta. This effectively makes each display an island that acts as though it were a single display as far as standard mouse traversal is concerned.

This configuration is useful for full-screen video-only output displays; for example, preview displays in video editing applications, or projected displays in realtime video effects applications. In both of these setups, the cursor is never needed on the video display. In the former, the preview

video display is a distractor target on which the cursor may get lost. In the latter case, moving it there accidentally may be detrimental as it would then show on the production output.

For users who wish to have explicit control over the ability to move the cursor from display to display, a small utility called MouseWarp was created for this thesis to augment the unreachable display mode. MouseWarp allows the user to *warp* the cursor between displays via a global hotkey. MouseWarp listens for a pair of system-wide key combinations, respectively jumping the cursor forward and backward between attached displays. This makes displays that were previously inaccessible by cursor movement reachable using the hotkey.

Windows, icons, and other GUI objects that are normally moved by dragging can still be transferred between unreachable displays with MouseWarp. This is accomplished by starting a drag (mouse down), invoking MouseWarp before the drag operation completes, and then completing the drag and drop (mouse up) on the warped-to display.

3.5.3.3 Bezel compensation

A visual side-effect of imposing a logical gap between two screens is the creation of a physical gap in GUI objects that span the displays, as illustrated in Figure 3.2. This may be used to account for bezels or other physical limitations that prevent two displays from being abutted as closely as their logical counterparts are [77]. Note that for large gaps between screens, a utility such as MouseWarp will be required to move the cursor and other GUI artifacts between displays. Adapting MouseWarp to warp based on cursor position rather than a key combination would allow screens separated by a substantial gap for bezel compensation purposes to be traversed as though no gap existed, if such a configuration were to be used as a standard extended desktop. This would be a rather straightforward modification to `IOHIDSystem::_setCursorPosition()` described in Section 3.5.3.1.

3.6 Summary

Using the Reflect multi-display configuration utility provides additional flexibility to augment the standard mirror and extended desktop modes on Mac OS X. This was achieved by patching a system file to allow secondary screens to be placed anywhere in global screen space, rather than being constrained to abut or mirror an existing screen. Coupled with MouseWarp, these tools offer additional functionality for common multi-display tasks, as well as addressing the needs of several

specialist applications. This functionality is summarized in Table 3.2.

Table 3.2: Summary of multi-display modes, visual and control usage

Mode	**Visual effect**	**Control effect**
Mirror/Clone[a]	Duplicates existing display	n/a
Extended Desktop	Expands desktop area	Cursor travels between displays where edges abut
Disjoint	Provides bezel compensation (Section 3.5.3.3)	Display edge sticking (Section 3.5.3.1), Unreachable displays (Section 3.5.3.2)
Overlap	Projected display feathering (Section 3.5.2)	Cursor may exist on two displays at once (Section 3.5.2)
Inset[b]	Mirror subregions, Zoom a subregion, Provide privacy (Section 3.5.1)	n/a

[a] Mirror/Clone is a special case of Inset
[b] Inset is a special case of Overlap

The first two modes (mirror/clone and extended desktop) are standard on PCs. The other three are not, but are provided by Reflect. The most general new mode is overlap, which has inset as a special case and that in turn has standard mirror/clone as an even more specialized case.

In addition to the usage covered in this chapter, the Reflect multi-display configuration utility provides a platform through which new uses for these modes may be discovered.

Chapter 4

An Ideal MDE Device Architecture

In this chapter, the requirements for an ideal MDE device architecture are developed. The chapter begins with a historical perspective and a discussion of the difficulties encountered in current MDEs. This background informs the requirements for an ideal MDE device, which are presented in two parts. For each high-level requirement a broad recommendation is made, which assumes unlimited development resources. In addition, a low-level *next action* is suggested, which is a scaled back requirement that may be the basis of a prototype implementation. In Chapter 5, this set of low-level requirements is gathered and used to develop a prototype implementation.

4.1 Display History

In an oft-cited 1968 paper, Ivan Sutherland (widely acknowledged as the *father of computer graphics*) and Ted Myer summarized the state of the art for high-performance computer graphics processors [52]. They observed a 'wheel of reincarnation' phenomenon that they characterized as the tendency of engineers to add functionality to a graphics system until it was no longer cost effective, and to then split the system into smaller pieces with a low-cost display component attached to a higher-cost general processor. Beginning again with the low-cost display component, this process is potentially repeated *ad infinitum*.

At the time the Myer and Sutherland article was written, computer graphics had just completed one revolution around the wheel of reincarnation. The result was very powerful display 'channels' for line drawing (vector) graphics that had built-in matrix multiplication, subroutines, and limited forms of conditional testing. These were used to drive CRT displays, each of which was only slightly more capable than the first computer graphics displays that had appeared a decade or so earlier. The division of labour was decided in part by the need to process large amounts of 3D information to update a dynamic display, and the desire to offload interactive operations such as hit testing and cursor management from the main computer.

Every advance in display technology seems to recapitulate the experience described by Myer

and Sutherland. Not long after their article, bit-map (raster) graphics began to emerge as an alternative to vector graphics. By the early 1980s there were many raster graphics displays available, but only a few had the power of the older-style vector graphics display processors. This changed dramatically over the next decade as VLSI allowed for much faster and cheaper implementations of the functionality required for full 3D graphics. Today, it is taken for granted that every personal computer will have full 3D colour graphics with at least 1280×1024 resolution on a built-in display. Furthermore, it will have the capability to simultaneously drive an external display at comparable resolutions. It is worth examining the distribution of functionality within the current display subsystem to gain some insight into where we may next see ourselves as we continue going around the wheel of reincarnation, and how this may play out with respect to multi-display environments.

4.2 Intelligent Satellites

In 1974, Van Dam *et al.* continued around the wheel of reincarnation by describing a move away from the mainframe and terminal paradigm popular at the time, towards a future where terminals would become *satellites*, or general purpose computers in their own right [94]. In the context of graphics systems, these satellites would offload to the terminal some of the processing minutiæ such as clipping or event handling, lowering the load on communications channels and the mainframes themselves, as well as provide additional functionality beyond that of a simple terminal. In the intervening years, this vision has been realized in a manner far beyond what Van Dam *et al.* anticipated, to the point where the satellites have eclipsed mainframes altogether and complete computers are packaged in a device the size of a book.

Thirty years later, the concept of satellite graphics hardware is being demonstrated again—this time at the level of the graphics subsystem within a computer—as programmability is added to the previously fixed-function pipelines of a Graphics Processing Unit (GPU). This is shown in Figure 4.1. Initially intended to support 3D graphics processing tasks, the parallel processing capabilities of the GPU have allowed GPUs to take their place in the wheel of reincarnation: they are already being used for non-graphics tasks such as physics computations [32] and general purpose stream processing [44, 16].

Van Dam *et al.* argued that while some processing may be offloaded from a mainframe towards a satellite computer, there would always be a need for the superior processing power of a larger system [94]. As predicted by Moore's Law, the chips providing the processing power have shrunk

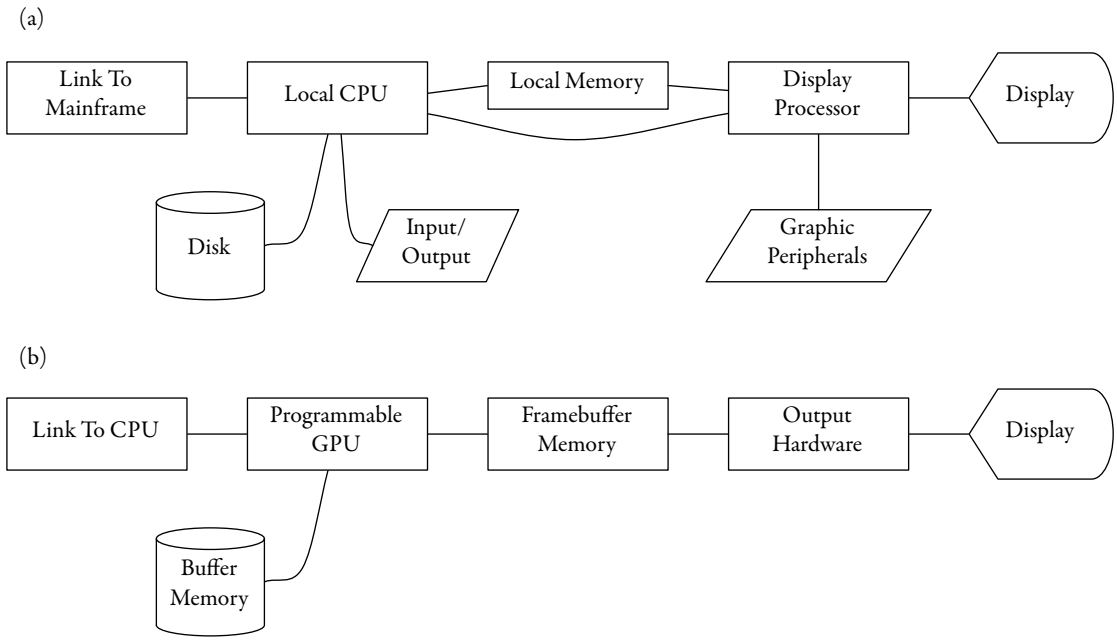

Figure 4.1: Graphics system block diagram. Mainframe / satellite system circa 1974 (a), GPU circa 2006 (b).

to the point where this is often not true for the needs of personal computer users [51]. However, one thing that has not become any smaller—and is limited by human vision capabilities rather than technology—is display size. This is in fact growing.

4.3 Difficulties Encountered in Current MDEs

There are three broad areas in which current display technologies limit the abilities of MDEs. These are: connectivity, ease of use, and multi-computer capabilities. The following examples serve to illustrate these points.

Laptops are an extreme example of the size of a computer being limited by human vision capabilities rather than technological capability; the size of the entire device is often determined by the size of the display. This size is in turn determined by the simultaneous need for a large display to match the resolution of human vision, and for a small display to allow the device to fit in a briefcase or backpack. When used in a mobile situation where external infrastructure is missing, this on-board display is as much as can be hoped for. However, many laptops are used in environments such as offices or media rooms, which contain pre-existing external displays that may be connected to the laptops for the duration of their use in the environment. However, due to

physical constraints that also limit the number of graphics output ports (i.e., a single onboard display, and at most one further external display connection), the display capabilities of a laptop are limited. Even desktop computers are often bound by physical constraints on the number of display output ports they may contain, due to the finite number of available expansion slots suitable for graphics cards. This limitation affects the scope of the MDE with which a user may interact. The USB-based displays offered by Samsung are one solution to this issue, as they allow up to five displays to be connected to a single PC [64].

In addition to the number of displays that may be supported by a given computer, the accessibility of displays is also constricted in today's MDEs. In a static desktop configuration, there is little difficulty in directly connecting a display to a nearby computer. But in an environment where display surfaces are an implicit part of the landscape, distance and addressability can make it difficult to correctly associate a given display with a particular PC. For example, in an environment with two projectors and a flat panel display permanently installed, labelled ports may be provided via a wall jack. However, the jack may not be in a convenient location and the labels may be incorrect or not informative enough. A better solution would be to provide multiple jacks around the room, with all three devices available on one connection. Display device identification would occur on the computer, with text or even images describing each device. This requires an investment in physical infrastructure beyond what is usually available.

Multiple users are not well supported by current display technology. This problem extends to MDEs. Individual displays cannot easily be shared. In an environment with many small displays, this is less of an issue as each user can connect to their own display. But in an environment with a large shared display, multiple users should be able to share the display simultaneously, rather than consecutively. Even when the display is shared consecutively, the process is often slow and error prone, as each user must physically pass around and re-connect a video cable, and subsequently configure their PC for the new display.

One solution to these problems is to adopt the model suggested by Van Dam *et al.*, and once again push computing power further away from the CPU, towards the display device [94]. By placing general purpose computing hardware at the display, it may then be possible to place displays on existing communication buses such as FireWire, USB, or a TCP/IP network to aid with connectivity issues. In the process, updating the protocols that move data to display devices would increase their ease of use. Increasing computing power on the display and utilizing multi-point connectivity media may enable multi-PC and multi-user capability. This ideal MDE display device

can be considered a **satellite display**, named after the satellites introduced by Van Dam *et al.*. A brief sketch of this vision is illustrated in Figure 4.2.

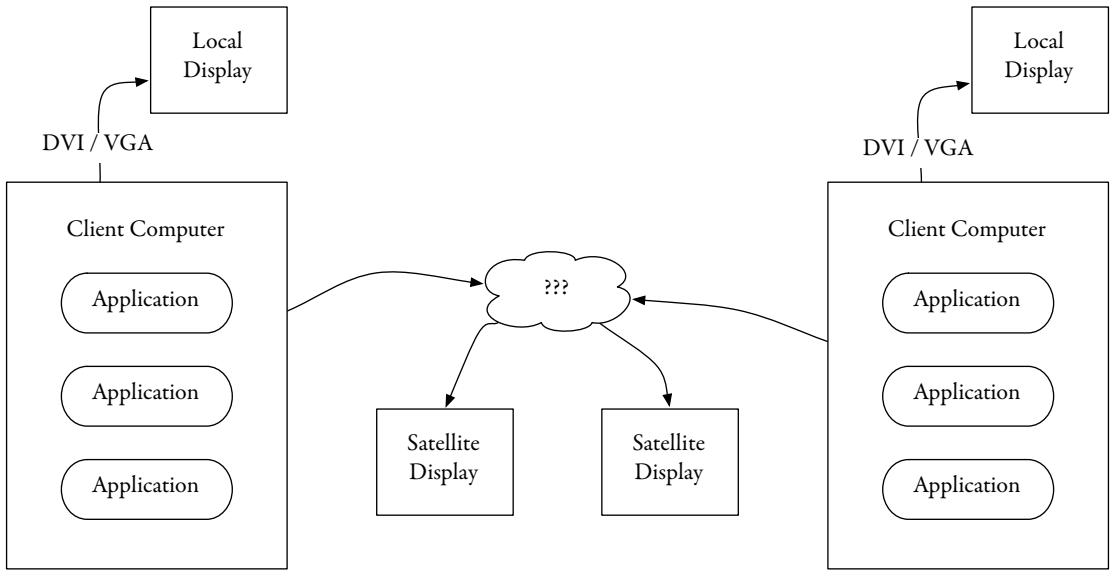

Figure 4.2: Proposed system overview. Each client computer is connected to a local display via a traditional DVI or VGA link. They are also simultaneously connected to multiple satellite displays through a mechanism that is left unspecified in this idealized view.

The remainder of this chapter will present nine requirements for an ideal satellite display. Each requirement includes a high-level or broad-reaching recommendation, which is then distilled into a low-level or technical action that can be implemented immediately as part of a prototype implementation of a satellite display. The details of a prototype satellite display implementation based on this ideal model will be covered in Chapter 5.

4.4 Requirements for an Ideal Satellite Display

This section presents nine requirements for an ideal satellite display. These are grouped into the following three broad areas: connectivity, ease of use, and multi-user capability.

4.4.1 Connectivity

As discussed in Section 4.3, display device connectivity is an issue faced by most current MDE users. This arises primarily because of the lack of output ports on the PC hardware itself which is due to a lack of physical space for expansion cards and connectors. In addition, current PC hardware is

not designed to be used with the next generation of embedded and ubiquitous display technology. This technology does not necessarily conform to the display size and resolution expectations that PC hardware manufacturers consider. To address these current limitations, we consider what happens if general purpose processing capabilities are added to the display device to allow multiple displays to be connected over a single bus, in addition to handling the details of low-level device management.

4.4.1.1 Requirement 1: single graphics port

Current graphics hardware is capable of driving multiple display devices from a single chipset. It is presently common for a single graphics chip to drive two devices, but this number is often derived from the physical inability to fit more than two output ports on a single expansion card rather than technical limitations of the graphics processor itself. Specialty graphics cards from manufacturers such as Matrox [46] can drive three or four displays from a single chip, or multiple chips can be utilized together on a single board. These solutions typically utilize a single high-density custom connector on the graphics card, with a breakout cable that splits the signals into three or four standard VGA or DVI connectors. An additional way to employ more displays with one desktop PC is to install multiple graphics expansion boards in one computer chassis.

While the above solutions may suffice for desktop computers, laptops are faced with a more significant shortage of connector space and at best typically offer one external display connector. Further, it is not practical to put a non-standard high density graphics port on a laptop and expect users to carry breakout cables with them in case they need to use more than one external display.

Recommendation: Redesign the graphics transport protocols to allow image data to be multiplexed over a single link, rather than requiring individual point-to-point streaming links between the source (the user's computer) and each sink (display device). Achieving this recommendation removes the physical limitations that prevent devices such as laptops, small multimedia computers, and similar devices from using multiple displays.

There are many network topologies that may be suitable for an alternative physical graphics link configuration. A distributed bus (examples of this topology include FireWire and USB) would work well for this application as this layout provides simple physical connectivity for the end user. Furthermore, it logically provides the same bandwidth to all devices, making it possible to guarantee performance across devices. A second option may be to use a star topology similar to current

IP-based networking configurations. Regardless of the topology chosen, Quality Of Service (QOS) channels should be provided for time-critical data such as cursor movement. To realize the goal of transmitting multiple streams of image data over a single graphics link without substantially increasing the link bandwidth, the data must be compressed. Several methods for compressing image data are discussed in Section 4.4.1.3.

Technical Action: Creating a single-port graphics bus would require a fundamental overhaul of the current computer graphics architecture. This requires significant engineering and manufacturing effort from all the vendors working in this market. In the meantime, existing IP-based networking technology provides a working single port addressable bus that may be used for prototyping purposes. As contemporary IP-based networks are not capable of providing comparable bandwidth to a single point-to-point graphics link, substantial image compression will be required to achieve reasonable performance.

The single graphics port bus proposed in this section would allow Scenario 3 from Chapter 1 to conclude differently:

Scenario 3 revisited, changes in bold

> It's been a good year for the company Alice works at, and there is a surplus in the equipment budget. Corporate IT has determined that using multiple displays offers a productivity boost, and decided that everyone working in Alice's office should have two displays. Alice wants to continue using her laptop as it contains all her files and applications, but it can only drive a single external display, in addition to the on-board LCD. **Alice connects her two new satellite displays to the network jack on her laptop by daisy-chaining them, enabling two external displays. If further budget windfalls allow additional satellite displays to be purchased, Alice may continue to plug them into a standard network switch attached to the port on her laptop.**

4.4.1.2 Requirement 2: display device homogeneity

As PCs increase in computing power, users are finding increasingly esoteric uses for devices which were once limited to the domain of office tasks. Personal computers have always been used for gaming, but this is now a primary activity for many users. This demand for gaming machines has

driven recent research and development activity with regards to the high performance multi-media uses of a computer. Subsequently, these innovations have been adopted for other purposes such as real-time rendering of video and graphics for live display purposes in video installations, displacing expensive and specialized video mixing hardware.

Coinciding with the improvements in graphics processing within the PC realm, static displays used for architectural, entertainment, and signage purposes are being replaced with digital panels which offer increased flexibility in configuration and dynamic content presentation [80]. As this area grows, content producers look to commodity PC hardware to drive the displays. Due to their embedded nature, such installed displays are often custom sized [25] and don't fit into the resolution requirements offered by conventional PC graphics hardware. For example, an LED marquee may have a resolution of 3000 × 100 pixels, far outside the aspect ratio and standard size of a current PC display.

In addition, the use of large, high-resolution tiled video walls is gaining popularity for visualization of large data sets [36]. Driving these high-resolution displays often requires custom hardware to manage the display tiling [99], but ultimately many researchers are generating imagery using PC computers with commodity graphics cards [88]. It would be advantageous to support the use of PC hardware with these display devices by allowing the fixed install hardware to handle the tiling and simulate a single large satellite display to PC users. Alternatively, by exposing each tile as a satellite display, the PC user could connect to and arrange the extended desktop tiling manually. This has the advantage of only requiring a single cable to do so, rather than one cable per tile.

By providing an open, lowest common denominator API to the low-level display subsystem, any networked device could be attached to a PC as though it were a standard display, as long as it is capable of interpreting a bitmap image. This may lower the entry cost to interface PCs with custom display hardware such as video walls [24, 9], small notification displays [50], and other low resolution media surfaces, as it would no longer be necessary to capture or re-digitize a conventional DVI or VGA video signal to drive the display hardware. In contrast, all current PC graphics interfaces have a minimum supported resolution of 640 × 480 pixels, which may not be suitable for these non PC-centric applications.

Recommendation: Provide flexibility to allow users to interface with oddly sized displays that are constrained by the environment, not a hardware engineer's notion of a computer display. Not all PCs are being used on a desk, supporting office applications.

Technical Action: Redesign the link protocols used by display devices such that they better support non-standard devices. Utilize existing networking media to uniformly deliver graphics data to non-standard devices, both small and large.

Developing a homogeneous display architecture as described in this section would allow Scenario 4 from Chapter 1 to conclude differently:

Scenario 4 revisited, changes in bold

> A university has installed a rear-projected high resolution tiled display for scientific visualization research. The dedicated computer cluster driving the display handles the tiling and scaling, and natively run high performance parallel rendering visualization software. **The lab technicians configure the rendering cluster to emulate a single satellite display which spans the entire extent of the tiled display. Visiting researchers then attach to this satellite display, and can integrate it seamlessly with the extended desktop configuration of their PCs (illustrated in Figure 4.3).**

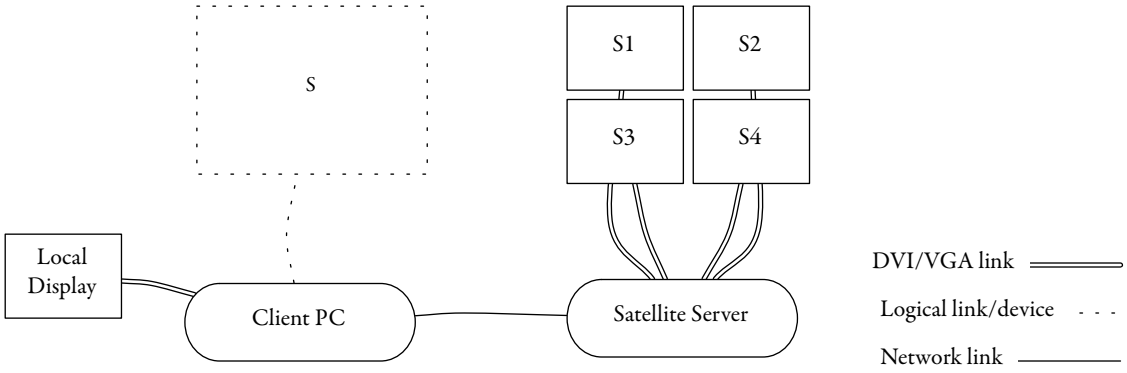

Figure 4.3: Logical display consolidation. In this example, a graphics cluster acts as a satellite display, tiling 4 displays (S1-S4) into one device that is presented to the client PC as one large logical display (S).

4.4.1.3 Requirement 3: On-display processing

The bandwidth required to transport graphics data between a graphics chipset and a display device is among the highest utilized anywhere within a personal computer, rivalling and often exceeding

that of the bus between the CPU and main memory. Although high performance from a computing perspective, the graphics communications protocols and transport channels such as VGA and DVI are simplistic, and employ brute force techniques to maintain acceptable visual performance levels. This architecture, born out of the analogue display technologies of the past, places the lowest demands on the display by not requiring any image storage on the display device itself. Instead, a full display's worth of bitmap data is sent from the computer to the display every time the display needs to be refreshed, typically at a rate of 60 Hz or higher. This refresh is required to retain an image on a CRT display, regardless of whether the image is static or in motion. In the case of a static image, the same data is sent repeatedly over the graphics link. The refresh frame rate of a display device should not be confused with the update frame rate of the source, which is the rate at which displayed images may change in order to be perceived by the human visual system as representing motion. A static image has an update frame rate of 0 Hz, while a moving image typically has an update frame rate of 15-60Hz.

Due to the relatively recent adoption of digital display devices and the deep-rooted analogue heritage of the graphics pipeline, there has been little incentive to optimize this approach. However, digital display devices behave differently than analogue displays, and these differences may be exploited to provide greater functionality with modest additional cost and engineering. By adding a small amount of general purpose processing to the display, the current point-to-point connector may be replaced with a more generic multi-point bus, allowing multiple displays to be plugged into a single port on a computer.

Whether or not a digital display technology is capable of storing and displaying its current image indefinitely, this capability is easily attained by adding a framebuffer to the device. This allows the graphics link between the computer and display to idle when the display image is not actively being updated. This functionality mirrors the analogue storage tube vector displays from the 1970s [8], once again recalling the wheel of reincarnation.

Recommendation: By lowering the bandwidth requirements of a single display, multiple displays' worth of image data may be transmitted over a single link without substantially increasing the link capacity beyond what is currently feasible. To reduce the bandwidth usage, several compression techniques may be used; a few examples are given here. First, modern GUIs generally consist of static images interrupted by short bursts of motion. For example, when using a word processor, the user interface is static unless the insertion point is blinking or the user is actively

typing. Even then, the frame rate of character insertion is much lower than the 60 Hz refresh rate used by current display links. The animated images that comprise a video stream typically have frame rates from 15-30 Hz; which is less than half of a display refresh rate, meaning that images must be sent at least twice per frame. Temporal compression may be implemented by only sending data when the user interface is updated; repeated images need not be resent simply to support any internal refresh mechanism the underlying display hardware may need. This will allow multiple displays that update at different times to efficiently share link bandwidth. Second, spatial compression may be achieved by only transmitting updated sub-regions of the display image, rather than the whole extent of the display. In addition, basic 2D graphics operations may be performed on the display rather than on the computer. These operations, such as region copying, moving, and filling, are currently handled by the GPU rather than the CPU; spatial compression may be gained by simply moving these operations even further away from the CPU and closer to the display. Third, multi-media video sources such as DVDs and movie files are already compressed to reduce storage and transmission requirements. The decompression that occurs during playback may be moved to the display device, minimizing the bandwidth required to transmit the media to the display. This approach mimics current PC hardware which has already moved this task from the CPU to the GPU for more efficient processing of MPEG2 and H.264 encoded content [60, 1]. The next generation copy-protection schemes such as HDCP require the display device to decrypt video content [23], so additionally decompressing the media on the display device poses no legal hurdles to this process.

High volume image data, such as that produced by full-screen computer games, may need to continue to be sent uncompressed over the graphics link. To accommodate these applications, a channel with QOS provisions may be set aside on the link to provide backwards compatible streaming capabilities. This channel would provide a finite amount of bandwidth, perhaps the equivalent of one single-link DVI connection. In this case, a compromise may be to allow one display to receive streamed graphics data while the remaining displays receive packetized image data updating at a lower frequency.

To maintain the illusion of responsiveness, the graphics link will provide an additional low-bandwidth QOS stream for cursor position information. The displays must support hardware cursors and hardware accelerated 2D operations, much like current graphics cards do. This will allow smooth cursor movement across all displays regardless of the bandwidth being used for packetized graphics update data.

Technical Action: Prototype a satellite display device with the following characteristics:

- On-board framebuffer to handle refresh needs of underlying physical technology, as necessary.
- Offscreen VRAM to allow caching of image data.
- GPU that supports minimal 2D graphics operations such as compositing, region moves, region copies, and region fills.
- Decompression engine to handle processing and display of compressed streams in popular formats. Start with types managed by current GPUs: MPEG2 and H.264.

Iteratively determine the appropriate level of on-display processing necessary to achieve the desired bandwidth reduction and interactive performance capabilities.

4.4.2 Usability

With the redesign of the display connectivity architecture presented in Section 4.4.1.1 and the addition of general purpose computing to the display in Section 4.4.1.3, it follows that these changes may be used for other purposes, such as the improvement of MDE usability. This includes increasing plug-n-play connectivity and making it easier to configure a display device at its native resolution. Regardless of the changes made, legacy software should continue to be supported without requiring recompilation.

4.4.2.1 Requirement 4: ease of connectivity

As laptops continue to gain market share in both the consumer and business markets, it is increasingly important to support their ease of integration with existing infrastructure. Home users want to be able to sit in their media room and use a laptop to control equipment or to share photos and videos on a large-screen display. Cost is a further motivating factor, as laptops become commodity items and the traditional TV set gives way to more expensive home-theatre-sized display devices. Similarly, business users are often called upon to set up and give presentations in unfamiliar meeting rooms, where they must use a shared projector or similar device.

Currently, accomplishing this often consists of explicitly connecting a video cable to the laptop, and then interacting directly with the laptop input surfaces. Ease of use difficulties primarily

stem from poor technical support; it is notoriously difficult to dynamically enable video output on Windows PC laptops (one often hears someone in the audience suggest 'try pressing function-F8 to switch display modes' when a presenter is having troubles connecting to a projector). Once the laptop has recognized the external device, the user still needs to adjust resolutions and configure mirroring or extended desktop mode. More often than not presenters simply do not bother, resulting in a blurry presentation in mirrored mode with the audience fully aware of the presenter's incoming email and instant message notifications. At least part of this situation could be improved with inset mirroring, as described in Chapter 3, Section 3.5.1.

The configuration of external video devices should behave in a manner similar to that of current wired and wireless networking: the computer dynamically and automatically detects the presence of networking resources, and protocols such as DHCP and ZeroConf issue the necessary information to successfully connect to the network and configure settings in an *ad hoc* way, allowing for immediate use. While some computers have automatic detection capabilities for display devices, they are not widespread enough for users to be able to depend on them. The display information protocol used to support this automatic configuration, EDID, simply lists all supported display resolutions (if it can: it is not yet able to correctly express common widescreen resolutions), but only some implementations specify which one is the native resolution of the display. Further information on the capabilities and limitations of EDID may be found in Section C.1 of Appendix C.

Recommendation: *Ad hoc* multiple display configuration should be as easy to accomplish as current *ad hoc* network configuration. When adding a display device to a Windows PC, there is still a shortfall between the current plug-and-pray behaviour and the desired plug-and-play.

Technical Action: Due to the complex hardware-based technology used to implement current graphics links (see Appendix C for details), it is difficult to experiment in this area without having the substantial resources of a display manufacturer. To circumvent this problem, a standard IP-based network could be used to prototype the attachment and configuration of display devices. Device discovery and configuration can then be handled using existing networking technologies such as DHCP and ZeroConf [20].

4.4.2.2 Requirement 5: visual fidelity

In the past, PCs primarily employed CRT technology for their display devices. Due to the analogue process by which a CRT forms an image, CRTs do not have a specific native resolution. Instead, they can support a wide range of resolutions and refresh rates. Component quality and electrical tolerances dictated an upper bound on the resolution that could be achieved while maintaining an acceptably flicker-free refresh rate, but in practice any lower resolution looked just as good. In contrast, the digital flat-panel displays used with current PC hardware have a specific native resolution mandated by the physical picture elements that make up the display. Attempting to support lower resolutions by scaling the input signal by non-integer ratios (such as scaling 800 × 600 up to 1280 × 1024) results in a blurry image. As the processing power and framebuffer memory of PC graphics chipsets has increased to support the gaming industry, they have caught up to and surpassed display technology at least in terms of the range of resolutions the chipsets support. However, due to the heritage of modern digital flat panels, the panels continue to support multiple scaled, non-native resolutions in an effort to emulate the old CRT displays of the past and maintain some measure of backwards compatibility. In addition, during this transition phase from analogue to digital display devices, some digital display panels also feature analogue (VGA) inputs for backwards compatibility. This introduces additional degradation to the video signal, as it must endure a conversion from digital to analogue, and then back to digital again.

Recommendation: Allowing a digital display to be driven at a non-native resolution should be a last resort, not the default behaviour. Due to the historical architecture in use to determine display resolutions on a PC, it is often not possible to determine the native resolution of a display without consulting the display's documentation. A technical explanation of this is provided in Section C.1 of Appendix C. Efforts should be made by the Video Electronics Standards Association (VESA) to remedy this, as it simply is not easy enough for end-users to optimally configure their digital flat-panel displays.

If two displays are being used in mirrored mode and they do not share a native resolution, other solutions could be considered that retain visual fidelity. Insetting a smaller screen on a larger display should be considered as a viable option, rather than always scaling it up to fit the larger display.

Technical Action: As part of the configuration protocol, displays should only advertise the native resolution of the device. In addition, whatever is possible should be done to maintain the temporal fidelity of the satellite display. Both image and temporal fidelity will have to be managed in concert with the compression techniques discussed in Section 4.4.1.3, in order to efficiently utilize the capabilities of the underlying network transport layer.

4.4.2.3 Requirement 6: software transparency

As stated in the preceding section, which emphasized minimal changes in usage between a real display and a satellite display, it is desirable to maintain the ability to utilize existing software whenever possible.

Recommendation: A satellite display should interoperate with existing PC software. Software should not be required to be re-built with a custom toolkit or framework.

Technical Action: Implement the satellite display system at a level such that existing application software operates correctly, without needing to be re-built. Some changes to the operating system may be necessary, but it is preferential to make these changes in a single place (such as a graphics driver) rather than for every application that may be run on the system.

4.4.3 Multiple Users

The networked display connectivity architecture presented in Section 4.4.1.1 and the addition of general purpose computing in Section 4.4.1.3 may also be used to give a satellite display support for multiple simultaneous users. The networked connectivity allows multiple devices to be connected simultaneously, while general purpose computing enables image data from those multiple sources to be composited on the display device in a meaningful way. This means that multiple users, each using one PC, may simultaneously share a single satellite display. Finally, because it is simply a display device, the data being shared on a satellite display is safe from being manipulated or digitally copied by other users.

4.4.3.1 Requirement 7: graphics compositing

Once computing capabilities have been added to a satellite display for data decompression and network management, this processing power may be utilized to support multiple users as well.

Specifically, this gives the display the ability to composite image data it receives, allowing the display to build up a screen image, instead of relying on a source computer to send an entire screen's worth of data. This may be done in two ways, depending on the capabilities of the source PC.

First, the source PC may composite a screen image, and simply mask out the GUI elements of interest before sending them. This is illustrated in Figure 4.4, where only window objects have been masked and sent. Note that the relationship between the windows is retained. This level of support is appropriate for operating systems with framebuffer window servers, such as Microsoft Windows XP and earlier, and Mac OS 9 and earlier. Recall that a framebuffer window server draws graphics directly to the framebuffer using a back-to-front painter's algorithm. This approach implies that once a region has been occluded by a higher object, the underlying region is no longer available.

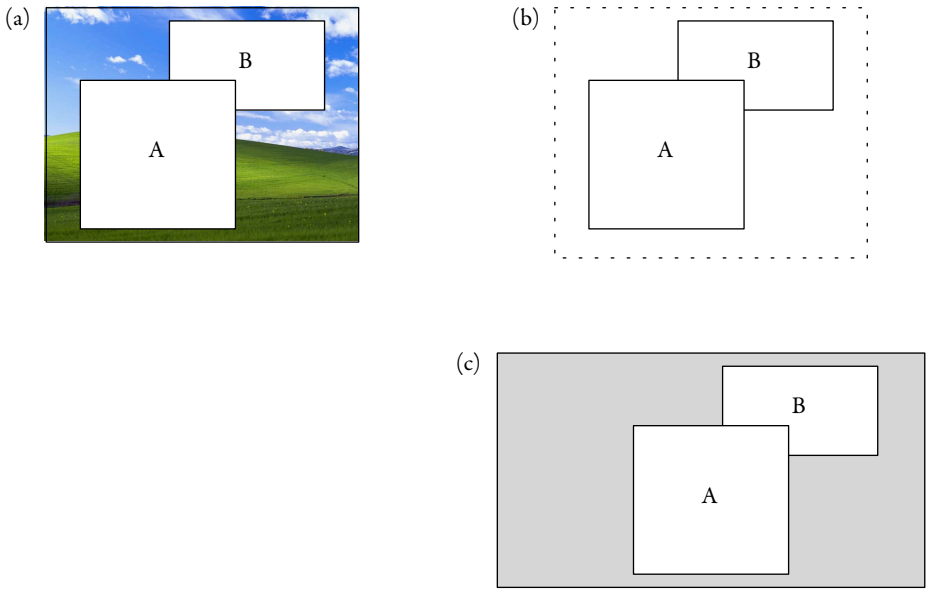

Figure 4.4: Masked GUI object transmission. With a standard unmasked DVI or VGA display, a source PC is required to send an image representing an entire screen, including the desktop background (a). Because a satellite display may composite GUI objects on the display itself, it can receive and process more complex graphics data. This is shown by the masked windows being sent in (b). In this example, the masked window shape is simply drawn onto a grey background to produce the final output from the satellite display (c).

Second, the source PC may send each GUI object as a discrete region, and allow the satellite display to composite its own screen image. This is shown in Figure 4.5, where individual window objects are sent, and the satellite display handles compositing everything together. Note that there is an arbitrary relationship between the window objects on the satellite display, and they do not

necessarily correspond to the relationships on the source PCs. This satellite display compositing method is available to source PCs that use a compositing window server, such as Mac OS X and Microsoft Windows Vista. Recall that on a compositing window server, each heavyweight GUI object is rendered entirely in off-screen VRAM at which point the image regions are assembled in the correct order to make up a screen image. Using a satellite display, the compositing step can be moved from the graphics board to the display device. A hybrid approach could perform some compositing on the PC, but other compositing on the satellite display.

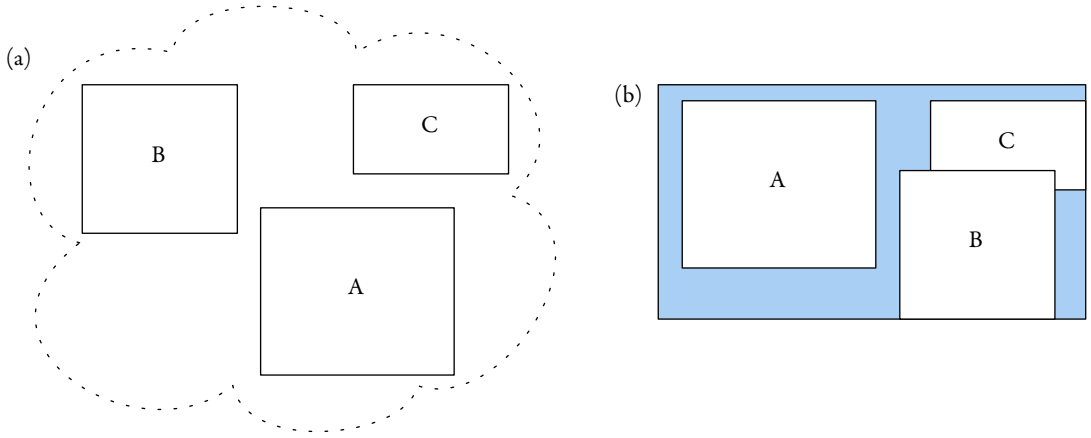

Figure 4.5: Fine-grained GUI object transmission. The GUI objects that make up a screen may be sent individually (a), and all of the compositing is done on the satellite display (b).

Processing on the satellite display is equivalent for both the masked and uncomposited GUI objects, as the satellite merely composites the regions it receives. Thus the satellite display may receive data from either type of source PC with no change. In addition, the satellite display may provide its own data sources to composite. For example, the display may have a TV tuner built in, or clock or calendar widgets. By compositing GUI objects on top of a TV image, the familiar picture-in-picture operation of a standard TV may be simulated.

Recommendation: Moving GUI object compositing duties to the display device affords flexibility in the way a display may be used. Simple features, such as providing a TV stream as desktop wallpaper, are straightforward to implement with this system and require little co-ordination between components. On the other hand, supporting multiple source PCs, while technically straightforward from a compositing standpoint, offers a host of usability issues. For example, it is unclear

what should occur when a user clicks on another user's window. In an MDE, it makes sense for this event to be forwarded to the appropriate PC, but this adds unnecessary complexity to the satellite display design. An ideal system design would allow an input redirection service to work in concert with a satellite display to provide this feature. Further complications arise when considering how to manage the space on a satellite display. As each PC independently recognizes a satellite display as though it were a local display, there is nothing preventing two windows from being placed in the same location. The satellite display then must decide on a stacking order, potentially obscuring one window. The appropriate semantics for the user of the hidden window to rescue their window are unclear.

Technical Action: Implement basic satellite display compositing operations now. Iteratively evaluate multi-user usage to determine the level of conflict resolution support necessary. Many problems are likely to be solved with social protocols, but explicit support for manipulating remote GUI objects may be necessary.

Compositing GUI elements on a satellite display as described in this section would allow Scenario 2 from Chapter 1 to conclude differently:

Scenario 2 revisited, changes in bold

> Bob sits at his desk, which houses one each of a Mac OS X and Windows PC, each with its own display. As a web developer, Bob authors content on the Mac, and uses browsers on both the Mac and Windows PCs to check cross-platform compatibility. In addition, he runs an email client on the Windows PC, and an instant messaging application on the Mac. **Utilizing a single satellite display as the primary display for his Windows PC, and connecting to the satellite display as a secondary display for his Macintosh, Bob is able to share the screen space on the *Windows display*, while preserving the entire *Macintosh display* for his content authoring application. He moves the Mac web browser and instant messaging client to unused space on the PC display, while still being able to see the PC browser and email client on that display as well. This is illustrated in Figure 4.6.**

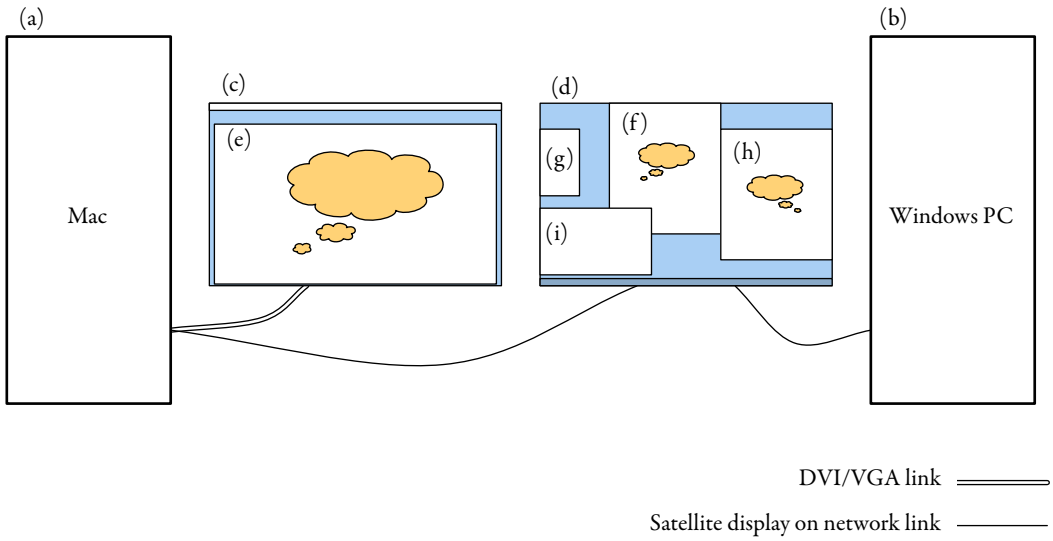

Figure 4.6: A single user with multiple computers and displays. The Mac (a) is directly connected to a local display (c), and the satellite display (d) is used as an extended desktop. The Windows PC (b) uses the satellite display as a primary display. The user authors content on the Mac (e), and tests it on browsers on both the Mac (f) and Windows (h) PCs. In addition, the user runs an instant message application on the Mac (g) and an email client on the Windows PC (i). These last four applications are shown on the satellite display, regardless of the computer they are running on.

4.4.3.2 Requirement 8: multiple simultaneous users

An environment with multiple displays is likely to have multiple simultaneous users that must be accommodated. Desktop PC-based operating systems make no attempt to solve this problem on a single machine; two users simultaneously typing on two keyboards will result in multiplexed gibberish, as there is no facility for multiple insertion points and multiply focussed windows. Using multiple distinct computers behind the scenes to drive an MDE solves this problem to some degree, as long as each computer has at most one user controlling it at any given time.

Given the tight dependence on a mouse cursor to provide an insertion point for keyboard input, a reasonable constraint is to create logical pairs of mice and keyboards, or *input channels* [89]. As operating systems continue to become more multitasking-friendly, support for multiple insertion points and simultaneously focussed windows is slowly coming within reach. This is achieved in part as GUI modality is phased out. For example, an open dialog box in one application no longer prevents the user from interacting with another application.

Until multiple input is a pervasive feature of desktop operating systems, satellite displays may

help simulate seamless MDEs by providing display output flexibility to multiple PCs. Each PC will still be used as an input channel, but rather than being restricted to output on a dedicated display per PC, the output may be composited to one or more satellite displays.

Recommendation: MDEs should support arbitrary multi-user input and multi-display output capabilities. This would allow the users of an MDE to control any application on any display from any input channel. We have already considered some of the input redirection necessary to simulate and explore multi-user input with the Mighty Mouse project [15]. The satellite displays presented in this thesis complement Mighty Mouse by addressing multi-display output.

Technical Action: Utilize the compositing capabilities of a satellite display to simulate the ability for multiple users to share one display in an MDE. By adding all of the satellite displays in an MDE to a user's PC, the user is granted seamless access to the entire environment simultaneously with other users who are using the space. Because the satellite display is simply an output device, there is no mechanism to allow users to interoperate with each other's GUI objects. Interoperative functionality may be achieved by using an input redirection system in parallel with the satellite display.

Compositing GUI elements on a satellite display as described in this section would allow Scenarios 5 and 6 from Chapter 1 to conclude differently:

Scenario 5 revisited, changes in bold

> Five co-workers share an open plan office with an adjoining meeting area and a large shared **satellite display** projection screen for giving presentations. Three of them are working on the latest product design, and they quickly share with one another snapshots of what they have accomplished. **The three co-workers accomplish this by adding the satellite display-based projector to their workstations as part of an extended desktop. When they wish to brainstorm together, they may each drag windows on and off the projected display as though it were a personal extended desktop, and the satellite display handles compositing the windows from different sources for final output. This allows multiple users to share the projected display simultaneously, allowing for quick visual comparisons between documents on different PCs.**

Scenario 6 revisited, changes in bold

> Carol and Dave are in their living room, planning a vacation on their laptops, and watching TV on a flat panel **satellite display with a built-in TV tuner**. As they find flight or lodging information on the Internet, they **drag windows from their laptops to the satellite display to compare prices and schedules. As the TV content continues to play, the computer windows are displayed on top of the video presentation, emulating what is popularly known in the home entertainment market as *picture-in-picture*. This is illustrated in Figure 4.7.**

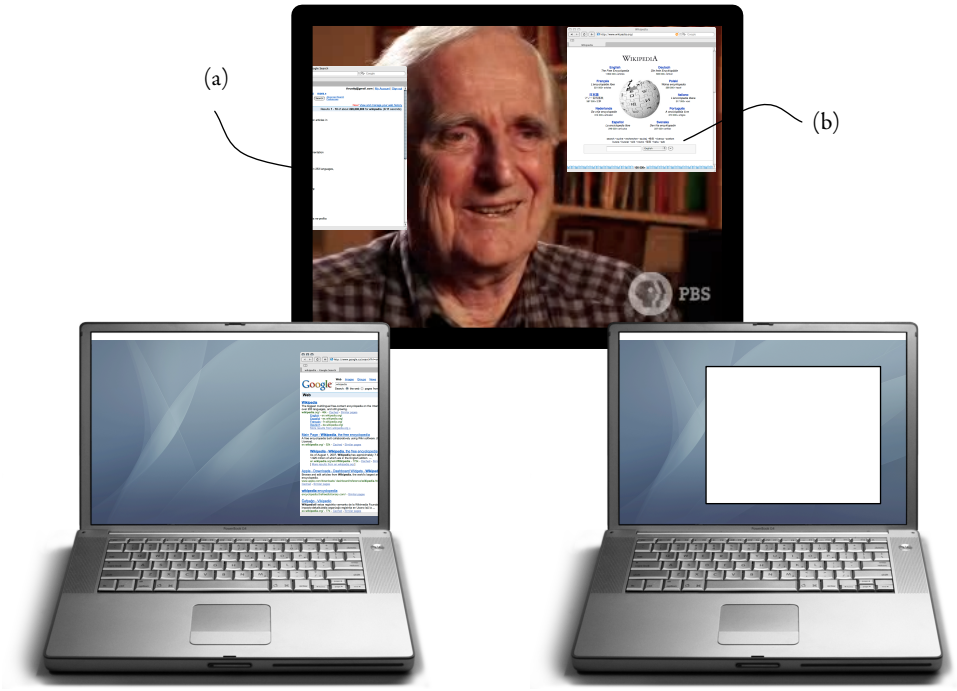

Figure 4.7: Sharing content on a home entertainment display. In this example, the laptop on the left has a window partially shared onto the TV (a), and the laptop on the right has a fully shared window (b).

4.4.3.3 Requirement 9: privacy and security

All of the flexibility discussed thus far comes at a cost. While the computing resources native to an MDE may be considered as a single unit, bringing a personal laptop into the environment changes things dramatically. For example, if the laptop is used as a control surface for the existing system,

should it also act as a display surface? Can other users of the system then view or alter data on the laptop?

Solutions proposed for this mixed environment range from being able to divide portions of the interface to limit what is displayed [38], to obscuring portions of the displayed objects [13].

Recommendation: As privacy requirements change from environment to environment, a plug-in architecture or similarly flexible framework may be utilized to allow the privacy of an MDE to be tailored to the specific application. In this way, privacy concerns may be explored iteratively to determine the best fit.

Technical Action: For an initial prototype, prevent input redirection between PCs participating in the MDE. This allows the satellite displays to be used as shared output devices, but does not allow multiple users to interact with each other's data.

4.5 Summary

This chapter has described problems that current display devices exhibit when used within an MDE. These include connectivity, ease of use, and a lack of multi-user support. To solve these problems, a new device was designed, called a **satellite display**. The satellite display addresses the MDE problems as follows:

Connectivity: To allow multiple displays to be connected to a single PC, the hardware transport layer was changed from a point-to-point graphics link to a multi-point network or bus (Requirement 1). Making this change required the addition of some on-display general purpose processing capability (Requirement 3). Because of these changes, a satellite display may also easily represent different display topologies such as tiled displays, or displays with non-standard sizes or resolutions (Requirement 2).

Ease of use: With the replacement of the physical connection type for connectivity reasons, new graphics transport protocols need to be designed. These protocols address ease of use issues such as proper plug-n-play device discovery (Requirement 4), and configuring a PC to use the native display resolution of a satellite display (Requirement 5). A requirement was added to ensure

that satellite displays would be able to seamlessly replace existing point-to-point displays without requiring legacy software to be modified to run on the new displays (Requirement 6).

Multi-user support: The proposed general purpose computing and networked connectivity architecture enables a single satellite display to support multiple simultaneous inputs. This is accomplished by compositing multiple video streams together on the satellite display itself, rather than on the source PCs (Requirement 7). As long as each user has his or her own PC, this allows multiple users to simultaneously share a satellite display (Requirement 8). Digital privacy is inherently supported due to the output-only nature of the satellite display–there is no way for a user to manipulate or copy the contents of another user's data through the display (Requirement 9).

As the requirements for an ideal satellite display were developed, solutions for each requirement were suggested, along with a low-level or technical action that could be implemented immediately as part of a prototype implementation. Our trip around Myer and Sutherland's wheel of reincarnation is continued in Chapter 5 with a description of a prototype satellite display implementation based on the recommendations made in this chapter.

Chapter 5

SDE: A Prototype Satellite Display Implementation

For this thesis, a prototype satellite display was implemented based on the requirements set forth in Chapter 4. These requirements are first reiterated and clarified in Section 5.1. This is followed by in-depth implementation details in Section 5.2. Next, usage of the prototype satellite display is described in Sections 5.3-5.5. Finally, the prototype implementation is compared to the requirements for both the ideal satellite display and proposed prototype from Chapter 4 to determine how well the goals were met.

5.1 Design

The design of an ideal satellite display was laid out in Chapter 4 with three primary requirements: improved connectivity over existing display devices, improved ease of use, and basic multi-user support. Those criteria were then distilled down to the following requirements for a prototype implementation.

- Connectivity:

  Requirement 1 Require only one hardware port on the PC for an arbitrary number of satellite display devices (Section 4.4.1.1).

  Requirement 2 Redesign existing link protocols to support satellite display devices with arbitrarily large or small device resolutions (Section 4.4.1.2).

  Requirement 3 Include computational capabilities in the satellite display, including basic framebuffer memory and a GPU that supports minimal 2D graphics operations (Section 4.4.1.3).

- Usability:

  Requirement 4 Enable ease of device discovery and configuration using existing networking technologies such as DHCP and the ZeroConf service discovery protocol [20] (Section 4.4.2.1).

Requirement 5 Maintain visual fidelity by only advertising the native resolution of the satellite display (Section 4.4.2.2).

Requirement 6 Maximize compatibility by ensuring that existing legacy software *just works*, and does not need to be recompiled or specially built to support the satellite display (Section 4.4.2.3).

- Multiple users:

Requirement 7 Allow multiple source PCs to simultaneously use a single satellite display by compositing the final output image on the satellite display device (Section 4.4.3.1).

Requirement 8 Enable multi-user support by allowing each user to connect to a satellite display with their own PC (Section 4.4.3.2).

Requirement 9 Implement privacy measures by only allowing connected users to manipulate their own GUI objects, not those of other users (Section 4.4.3.3).

In addition to the requirements defined in Chapter 4, three additional technical requirements are specified here to help manage the overall complexity of the system.

- Technical:

Requirement 10 Determine the feasibility of creating a prototype satellite display device from contemporary hardware, software, and available development resources.

Requirement 11 Do not allow display forwarding, i.e., a satellite display may not act as a client for a further satellite display. This prevents feedback loops from forming in the graph of display connections.

Requirement 12 Do not provide input redirection services between clients sharing a satellite display.

These requirements will be noted throughout the remainder of the chapter as they are satisfied (or not) by aspects of the prototype implementation. Note that Requirement 9 is subsumed by Requirement 12.

5.2 Implementation

Ideally, a satellite display device would be implemented in hardware. The desktop version would look like a regular computer display with a single multi-point-capable port (or internal wireless equivalent) to replace the current DVI or VGA graphics interface. Examples of similar devices are the Samsung 940UX USB display [64], and many networked video displays [80]. For prototyping purposes, however, a satellite display device was mocked up using a regular LCD panel, with the addition of a small form factor PC to provide the necessary computational support. Because the prototype implementation is an approximation of a real satellite display, it was called a Satellite Display Emulator (SDE). The development SDE is illustrated in Figure 5.1.

Figure 5.1: Prototype satellite display. The Mac Mini computer used to prototype the graphics and networking functionality is small enough to fit within the frame of the LCD panel. There are three cables used to connect the prototype satellite display: power for the display, power for the PC, and Cat-5 networking.

The remainder of this section goes into greater depth on each piece of the implementation. A general overview is given first, followed by a discussion of the hardware used to construct the SDE. Sections 5.2.3-5.2.5.1 detail portions of interest specific to each software component that was developed. Networking and other shared software concerns are covered in subsequent sections. The source code for all of the software developed for the SDE is available online. Details are available

in Appendix D.

5.2.1 Implementation overview

The SDE is made up of both hardware and software, with the bulk of the complexity being handled in software. The hardware consists of standard PC equipment, while the software is divided further into three main components: a **SDE Client**, a **SDE Server**, and a **proxy framebuffer**. Each of these components will be covered in detail throughout the rest of this section. The SDE was implemented using Mac OS X due to the author's familiarity with that operating system and the availability of necessary APIs.

Figure 5.2 illustrates the relationship between the components of the SDE software and associated key OS features. At a high level, the interaction between these components is as follows.

On the client PC, a proxy framebuffer provides a surface for the operating system to draw into, at the lowest level possible in the system. Because an SDE does not have any traditional display hardware directly attached to the client PC, the client operating system must be tricked into believing that a traditional display does indeed exist. By implementing the proxy framebuffer at this depth within the operating system, the remainder of the operating system and user applications treat the framebuffer surface as though it were just another screen—no further modifications of the operating system or applications are necessary to use them with an SDE. The proxy framebuffer is provided by a proxy graphics card–a piece of virtual hardware that does not exist in the traditional manner.

The SDE Client software also runs on the user's PC, acting as a driver for the SDE. The client monitors the window server, watching for cursor movement or window update activity in the region defined by the proxy framebuffer. When one of these events is detected, the client captures and encapsulates the event, and sends it over the network to the SDE.

Finally, the SDE Server software runs on the small form factor PC that is built into the SDE device. This software receives cursor and window information from the client, and outputs it to the display.

5.2.2 Prototype hardware

During development, the SDE was implemented with a small-form-factor PC paired with an LCD panel. The PC was a Mac Mini computer fitted with a 1.42 GHz PowerPC G4 processor and 1 GB

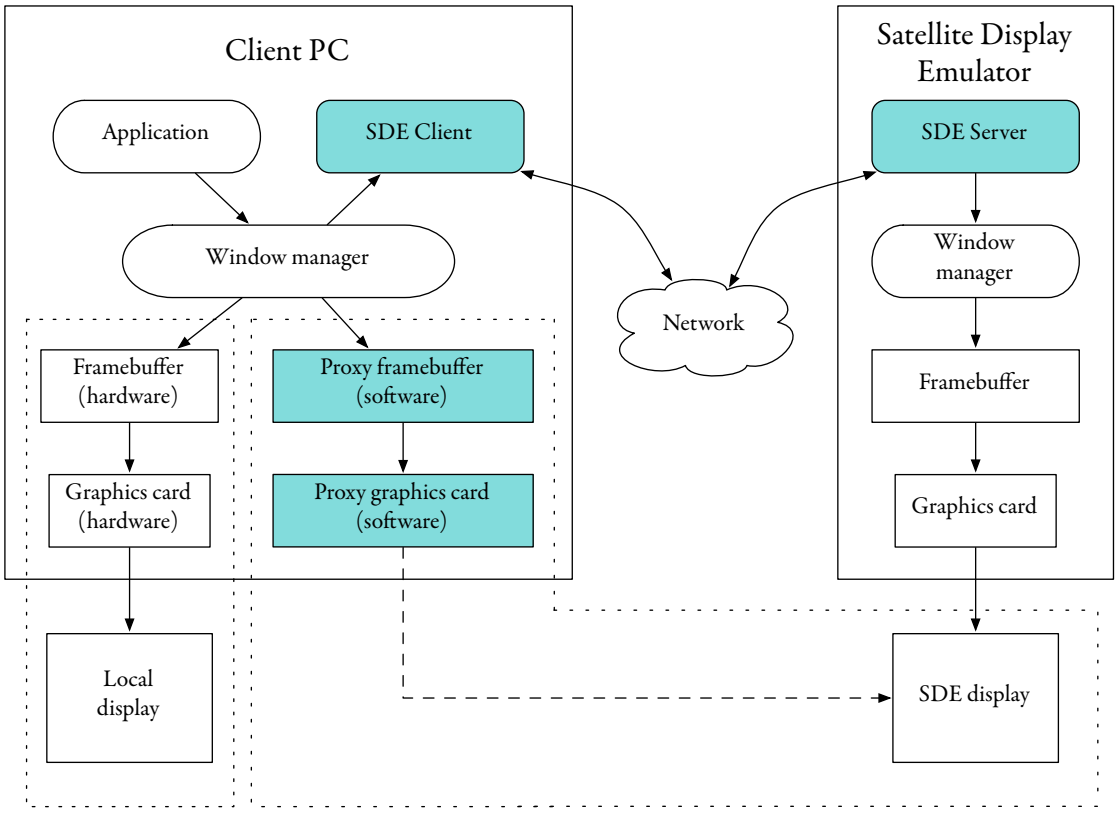

Figure 5.2: The SDE architectural overview. The client PC on the left has a real display attached to a real graphics card, these components are indicated in the dashed box containing the *Framebuffer*, *Graphics card*, and *Local display*. The client PC also contains a *Proxy framebuffer* and *Proxy graphics card*, which appear to the user to be connected to the display provided by the SDE. This is accomplished by the SDE *Client* software, which reads image data from the *Window server*, and transmits it to the SDE *Server*. Finally, the SDE *Server* software renders this data to the SDE output device.

of RAM, and the LCD was an 18″ NEC panel operating at a native resolution of 1280 x 1024. In addition to easing development by not requiring custom hardware design and fabrication, development time was cut dramatically, as the same networking and graphics APIs were used on both the source (user's PC) and sink (SDE) devices. Finally, implementing the SDE with a PC and high level software further allowed the use of existing applications on the display. For example, view-only utilities such as a clock or weather indicator may be left on the SDE regardless of whether or not source PCs are also connected. The Mac Mini-based SDE is depicted in Figure 5.1. For a more faithful reproduction of a hardware-only satellite display, a 20″ iMac fitted with a 2 GHz PowerPC G5 processor and 1 GB of RAM was also tested. This all-in-one form factor computer was

a close replica of a hardware satellite display when only the power and network connections were used.

The existing Ethernet port on the PC was chosen to receive graphics data from client PCs. This choice is acceptable as it enables multi-point communication, allowing multiple client PCs to connect simultaneously. This is needed to support Requirement 8. In addition, using either wired or wireless Ethernet as a transmission medium allows multiple satellite displays to connect to a client PC on a single physical port; this supports Requirement 1.

5.2.3 Proxy framebuffer

As mentioned in the overview, a proxy framebuffer is necessary to provide a surface on which the operating system can draw graphics intended to be output on the SDE.

The proxy framebuffer is provided by a proxy graphics card that the operating system can use as though it were real hardware outputting to a real display device. Rather than actively interpreting what is being drawn, the proxy framebuffer simply provides a surface in display space—the appropriate window contents are copied by the SDE Client from the window server with higher-level system calls. The relationship of the proxy framebuffer relative to the other components in the SDE system architecture is illustrated in Figure 5.2.

Because the SDE Client copies image data from the window server, rather than directly from the framebuffer, implementing a display surface at this depth is not necessary from a technical standpoint. With direct access to the operating system source code, a better method for creating a fake display would be to modify the window server, and allow it to create regions in display space with no backing hardware. However, without substantial support from the operating system vendor, this is difficult or impossible. Although it is significantly more convoluted and less flexible than modifying the window server, constructing a proxy framebuffer is within the means of a 3rd party developer, and allows for a prototype implementation of the concept.

Although the proxy framebuffer faithfully reproduces a raw framebuffer, it does not implement 2D acceleration or 3D hardware rendering. While the goal was to make the framebuffer as transparent as possible to 3rd party applications, it does not achieve this for graphics intensive applications. In this case, the applications either do not work, or do not work well.

In addition, the lack of real hardware backing the proxy framebuffer can cause problems integrating it with the Mac OS X driver architecture. Specifically, because there is no hardware to generate interrupts, events such as device addition and removal are not possible to synthesize. As a

result, the number of false displays attached to a proxy videocard is not dynamically configurable, and requires a reboot to change.

Because the proxy framebuffer is a proof-of-concept workaround to functionality better provided within the window server, these limitations were a reasonable tradeoff.

Low-level implementation details for the IOProxyFramebuffer and IOProxyVideocard kernel extensions can be found in Appendix A.

5.2.4 SDE Client

Once a proxy framebuffer has provided a region in display space representing an SDE, the SDE Client captures the contents of windows moved into this region and transmits the image data over the network to a display server for final output. The position of the SDE Client within the overall SDE system architecture is shown in Figure 5.2.

5.2.4.1 Window capture

The key component of the SDE Client is an undocumented Mac OS X system call, `CGContextCopyWindowCaptureContentsToRect()`. This system call grabs a bitmap of an entire window regardless of its on-screen visibility. This means that it does not matter whether the window is partially obscured or fully hidden, a complete image may be retrieved. To accomplish this at interactive rates requires a compositing window server, which Mac OS X offers. This allows windows to be sent to the SDE in individual streams for compositing on the display (possibly among windows from another client PC). This satisfies Requirement 7, which requires a compositing window server to support multi-user use of the SDE.

The Mac OS X screen capture utility, `Grab.app`, exposed this functionality, and provided a starting point to discover how it is accomplished. To uncover the `CGContextCopyWindowCaptureContentsToRect()` system call, Grab.app was reverse engineered using techniques similar to those described in Appendix B.

An API call similar to `CGContextCopyWindowCaptureContentsToRect()` named `printwindow()` is available in Windows XP. Although this call can also capture a hidden or partially obscured window, it requires substantially more resources to do so. It cannot be called at interactive rates as the window to be captured must be re-rendered to a special off-screen context specifically for the purposes of the `printwindow()` call, rather than each window being

rendered in this manner by default.

To determine which windows need to be captured by the SDE Client, the system call `NSWindowList()` is first used to get a list of all of the windows on the system. Every GUI object is contained within a window, including the desktop background. Each window has a corresponding z-index, which indicates the depth at which it is drawn to the screen. Anything at or below the desktop level must be filtered out by its z-index, so it is are not captured and sent. Next, `CGSGetScreenRectForWindow()` is called to compute a frame rectangle for each window. Finally, this rectangle is intersected with the screen frame of a proxy framebuffer representing an SDE. If the rectangle overlaps a remote display, the window is captured and transmitted.

By interpreting the screen data captured from the window server, it is possible to minimize the bandwidth required to transmit the data. Static images do not need to be continually sent over the network, as the SDE is capable of maintaining a cached copy. In addition, windows that are simply being moved only need to have positional coordinates updated on the SDE, because the window contents have not changed. A pair of callbacks have been provided by the window server to help with this task. These callbacks notify application software when regions of a screen have been updated with new image data. The first callback (`CGScreenRegisterMoveCallback`) is executed when a region has moved, for example, if a window has been dragged. The second (`CGWaitForScreenUpdateRects`) is called if a region has changed, for example if an image has loaded in a web browser. Unfortunately the region-moved callback cannot be used by the SDE Client, as experimental evidence suggests that it relies on the existence of 2D hardware acceleration which the IOProxyFramebuffer does not support. The region-changed callback does work with the IOProxyFramebuffer, however, and from this information window-moved events may be computed.

When the screen representing an SDE is refreshed on the client PC, a region-updated callback is triggered and the appropriate event is computed from the captured window data. If a window has been dragged, this event is reduced from transferring a full bitmap to a smaller set of updated coordinates. If window contents have been updated, a bitmap representing the window is sent to the SDE.

5.2.4.2 Cursor capture

The implementation of a GUI cursor on most computer systems is a predecessor to the compositing window server architecture. To remain responsive, cursors must be guaranteed to update on

each screen refresh, regardless of the load a CPU is experiencing. Compositing a cursor onto the framebuffer, including asking applications to re-draw regions of their windows invalidated by the cursor image, is too much work for the mouse-moved interrupt completion routine to accomplish under all conditions. To solve this problem, the cursor is drawn by the graphics hardware. A cursor image is sent to the graphics hardware, along with its state, such as visibility and position. Each time the pointer hardware issues a mouse-moved interrupt, the operating system forwards the position information through to the graphics hardware, which is responsible for painting the cursor at its new location and restoring the image at the previous cursor location. Whenever the cursor image changes, for example from an arrow to a hand, a new image is sent to the graphics hardware.

One consequence of the hardware cursor architecture is a lack of API calls to determine the current cursor image in Mac OS X. Whether it is an intentional feature or a byproduct of further cursor-based optimizations is unclear: the cursor image does not even show up in screen captures. As a result, the cursor position can be determined and sent by the SDE Client; however, without a way to capture the cursor image on the client PC, the cursor shape that is drawn on an SDE will not change regardless of the appropriate shape for the context on the client PC.

5.2.5 SDE Server

The SDE Server software is a lightweight application that advertises itself on a LAN using the Zero-Conf service discovery protocol. After an SDE Client connects, the SDE Server simply accepts incoming video streams, creates windows to house them, and sizes and positions them appropriately on screen. The position of the SDE Server within the overall SDE system architecture is depicted in Figure 5.2.

The SDE Server software was initially planned to be implemented in Java, with the intention of delivering run-anywhere capability. This idea was specifically targeting Unix-based rendering clusters that drive large screen tiled displays. The intent was to export the tiled displays seamlessly as a single large display that could be quickly and easily accessed from any machine capable of running the SDE Client. Time restrictions and difficulties in developing a custom cross platform serialization protocol resulted in a Mac OS X-only implementation of the SDE Server, with a cross platform implementation left as future work.

When running on a machine with multiple displays, there are two modes of operation for the SDE Server. Figure 5.3 illustrates the first mode, where each attached display is advertised separately, and a subset of the total number of displays may be chosen in the client. In Figure 5.3,

the user has chosen to arrange the satellite displays on either side of the local client display. This configuration gives the client user the most flexibility in terms of display placement.

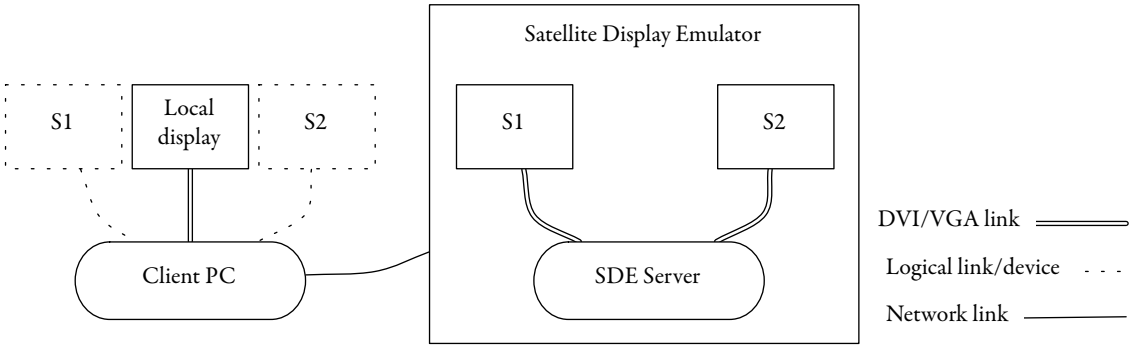

Figure 5.3: Individual display export. In this case, each display (S1, S2, solid outline) attached to the SDE Server is exported individually. The satellite displays may then be positioned independently on the client PC. This allows a user to create the arrangement shown at left, where a local display is inserted between two satellite displays (S1, S2, dashed outline).

If the SDE represents a large tiled display, the relationship between the tiles is already determined. Rather than requiring the client user to manually arrange multiple displays into the correct tiled configuration each time the SDE is used, the server software may pre-tile attached displays and export them as a single large framebuffer, as shown in Figure 5.4. This, coupled with the proxy framebuffer on the client, allows the client to use a single framebuffer that is much larger than what is possible with one local device. Because this mode has not been fully implemented, the upper limit of the number and resolution of the tiled displays that can be supported has not been determined. The SDE currently enumerates the displays connected to a multi-headed computer, but does not automatically find and advertise the largest rectangle that fits within the desktop area. An initial solution would be to manually provide this setting in a configuration file.

5.2.5.1 Window management

Because each window sent from a client PC is being captured in its entirety and sent as a discrete video stream, the compositing of client windows for final display can happen on the SDE. This is in contrast to VNC and other remote desktop tools, which scrape an entire screen and send it as a single pre-composited image.

The on-display compositing is not a difficult task compared to extracting un-composited video streams on the client PC. Further on-display processing such as the decompression of MPEG2 and

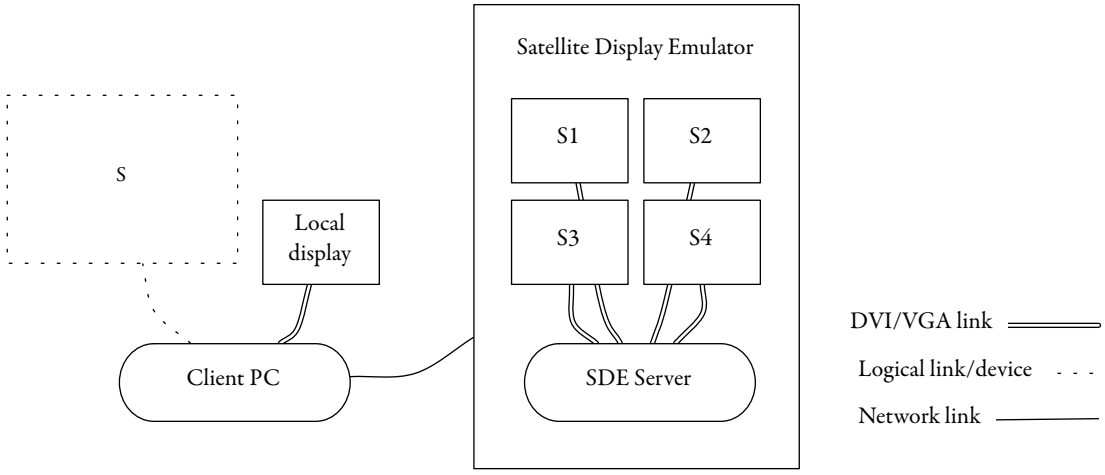

Figure 5.4: Tiled display export. In this example, the SDE pre-tiles four displays (S1-S4), and exports them as a single large display (S) to the client PC.

H.264 compressed video streams was not considered for the SDE, but thesee are obvious features to add.

The compositing that is done on the SDE partially satisfies Requirement 3. This feature allows windows from multiple client PCs to be simultaneously viewed on the SDE, satisfying Requirement 8.

5.2.5.2 Visual fidelity

The SDE Server only advertises the current resolution of the attached display device in accordance with Requirement 5. To achieve this, the SDE must be pre-configured to use the native resolution of the display device before the SDE Server is run.

5.2.6 Shared system components

Components in this section pertain to both the SDE Client and SDE Server. These include networking and software configuration.

5.2.6.1 Networking

To facilitate *ad hoc* satellite display usage, the network configuration must be effortless for the user. The ZeroConf service discovery protocol is designed to provide this facility between computers on the same Local Area Network (LAN) [20]. Because a satellite display is designed to be used in

a co-located environment, limiting network connectivity to a LAN is a reasonable requirement. Recall that a LAN may include both wired and wireless segments. By using the ZeroConf service discovery protocol, an SDE Client automatically sees all of the SDEs on a network, and the user may simply choose which one(s) to connect to.

A TCP network connection is used to transfer control and video data between SDE Client(s) and Server(s). A TCP connection was chosen to transmit video data to simplify the software design. The Cocoa Mac OS X object orientated frameworks allow objects to be easily transmitted over a TCP connection. Image objects may be transmitted directly in this manner, rather than requiring the design of a custom network protocol. For a real-time application such as the satellite display, UDP may offer higher performance, especially when coupled with custom image serialization code.

After a brief initial handshake, the client sends frames of video to the server. The handshake notifies the client PC of the native resolution of the SDE. Unlike the EDID [96] notification mechanism for DVI and VGA, there are no restrictions on the possible values that may be used to express the resolution of an SDE. This satisfies Requirement 2, and allows SDEs with non-standard resolutions to be connected.

For the prototype implementation, an extremely basic transport protocol was used. This protocol was based on NSArchiver, the built-in object serialization mechanism provided by the Cocoa programming framework. Selecting this method has one advantage and numerous drawbacks. While it enabled rapid development, NSArchiver is not portable beyond the Mac OS X environment, and is less efficient than protocols such as the remote framebuffer (RFB) protocol, which has been designed for this specific application. Although the RFB protocol is open-source, its structure does not lend itself well to being adapted to multi-window video streams. A new protocol based on RFB was designed, but implementation in both Objective-C and Java proved too time consuming for the scope of this research, and was subsequently abandoned.

5.2.6.2 Configuration

Because satellite displays are meant to be part of a very lightweight and *ad hoc* environment, minimal static configuration is required. The system automatically handles all network related configuration, and the participation of a particular computer is governed by its inclusion on the same LAN as the others. The current SDE implementation does not support password authentication or other fine-grained access control measures.

When the SDE Client application is started, it presents a list of available SDEs to which it can

connect, as shown in Figure 5.5. Initially, it was intended that the SDE Client would communicate the SDE resolution settings to the proxy graphics card, which would then reconfigure itself to accurately simulate the connected SDEs. As mentioned in Section 5.2.3, the prototype proxy graphics card does not have this capability. Instead, the user must manually configure the proxy framebuffer display resolutions to match the settings listed in the SDE Client connection window. Because of this, Requirement 4 is only partially satisfied.

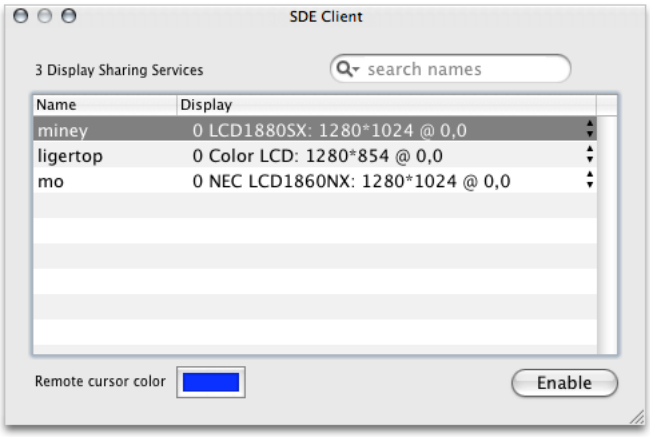

Figure 5.5: The SDE Client connection window. The first column lists available SDEs on the LAN. The second column provides information for each of the displays available on each SDE; this is intended to support the SDE Server split mode illustrated in Figure 5.3.

5.3 Single User Usage

The basic usage of an SDE for a single user may be summed up as follows:

1. Attach participating client PC and SDE(s) to a LAN.

2. Install the IOProxyVideoFamily suite of kernel extensions on the client PC, per the instructions in Appendix A.

3. Start the SDE Server software on the SDE(s).

4. Start the SDE Client software on client PC, and select and enable the desired remote display(s).

5. Ensure that the display resolution(s) of the proxy framebuffer(s) are equivalent to those advertised by the SDE(s). Position or mirror the SDE(s) using Reflect, or the system-provided `Displays.prefpane` configuration utility.

6. Use the SDE as though it were a display local to the client PC.

An example of an SDE being used in extended desktop mode is shown in Figure 5.6. The screenshot on the left represents the PC's local display, and the screenshot on the right is of the SDE. Note that by simply looking at this image, it is difficult to distinguish this configuration from a single screenshot taken on a traditional dual-head machine. However, these images were captured on a Mac Mini PC, which only has a single graphics output–without an SDE this configuration would be impossible.

(a) (b)

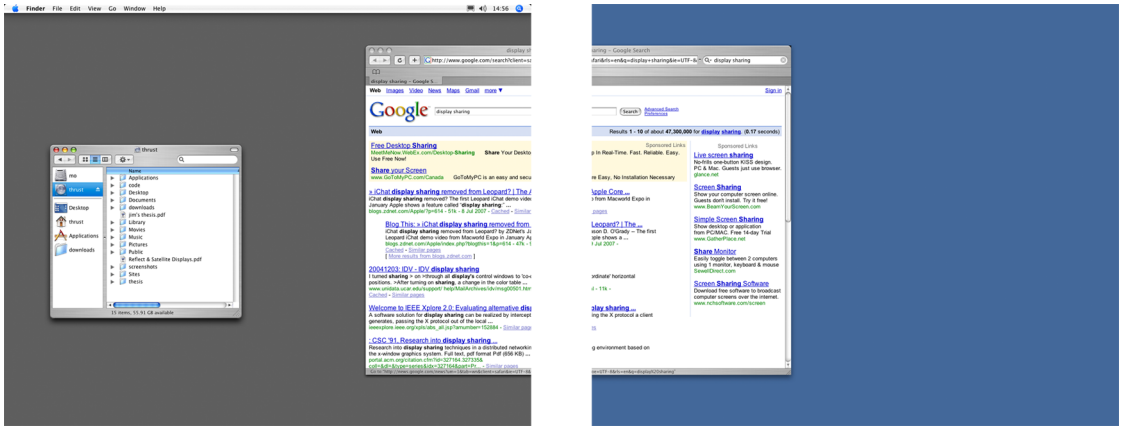

Figure 5.6: An SDE in use. The local display is on the left (a), and the output from the SDE on the right (b).

When multiple SDEs are connected to a LAN, they all broadcast their availability to SDE Clients that are open. Currently the SDE Client does not indicate to the user whether or not an SDE is in use by another client PC. Due to the multi-point nature of an Ethernet LAN, multiple users can each use their own SDE, but this requires the use of social protocols to ensure that users do not connect to other users' SDEs.

The SDE enables a subtle blend of single- and multi-user capabilities. Because the SDE is constructed from a regular PC and flat-panel display, it can locally run standard desktop applications. This is illustrated in Figure 5.7, where weather and clock widgets run on the SDE, regardless of whether or not a client PC is connected.

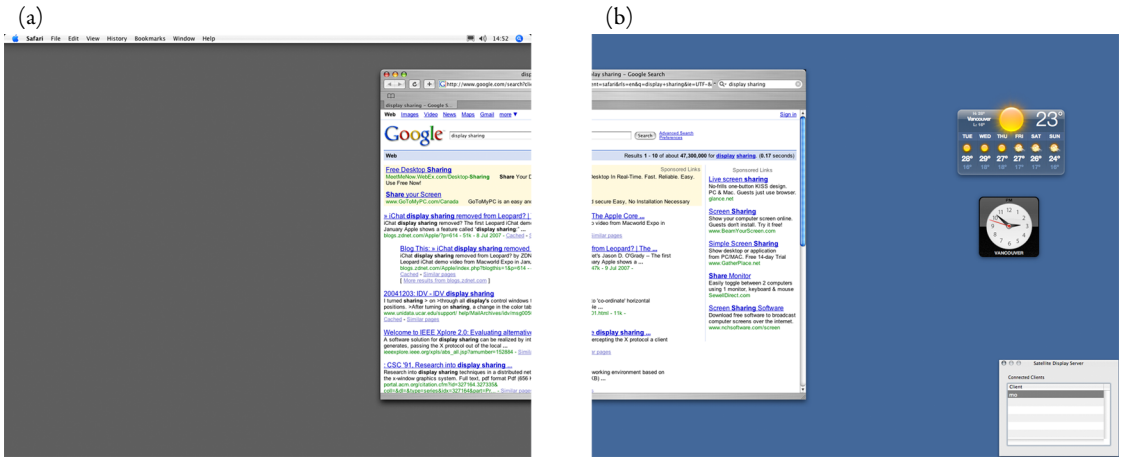

Figure 5.7: An SDE with local content. The local display is on the left (a), and the output from the SDE on the right (b). In this example, the SDE locally hosts a weather and clock widget, which remain on-screen regardless of whether a client PC is connected or not. In addition, a diagnostic window for the SDE Server software is shown at the bottom right; this lists the currently connected client PCs.

5.4 Multi-User Usage

Using an SDE with multiple users is accomplished in a manner remarkably similar to using it with a single client. Rather than checking whether other users are already connected to an SDE and avoiding displays in use, a user simply connects anyway. The result is that client windows are composited on the SDE, as shown in Figure 5.8. In this example, two users are sharing one SDE. Although in this instance all three displays align geometrically from left to right, this does not need to be the case. Both users could configure the SDE to be an extended desktop to the right of their local display: in this case, both users would access the SDE by dragging GUI objects to the right. This may be a more intuitive configuration if two users were sitting to the left of a large shared display. To both of these users, the shared display should be positioned to the right of their local displays.

In the current SDE implementation, there is no visual distinction between multiple client cursors on a shared SDE. One solution is to provide user configurable colour or shape settings.

5.4.1 Privacy

To both reduce the complexity of the SDE implementation and maintain some measure of privacy, user input events such as mouse clicks and key presses are not forwarded between clients sharing an SDE. Although multiple users can share an SDE as an output device, the collaborative capabilities

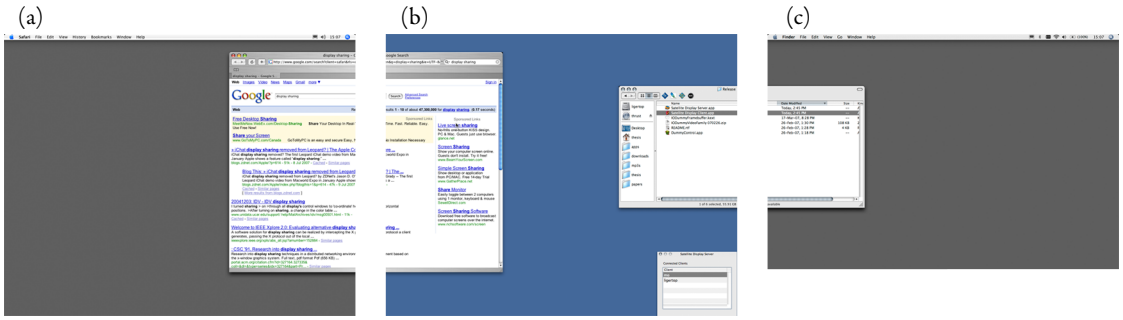

Figure 5.8: An SDE with multiple clients. The SDE is shown in the centre of the screenshot (b), with a client to each side (a, c).

stop there. If users wish to support further collaboration, such as being able to interact with shared GUI objects, the SDE system may be used in concert with an input redirection tool such as Mighty Mouse [15].

5.4.2 Usability

Simulating multiple-user support on what is essentially a single-user system is not without its drawbacks.

A usability quirk stems from the lack of input forwarding mentioned in Section 5.4.1. With many windows filling a shared SDE, it is easy to forget which windows are owned by another user, and therefore unavailable for manipulation. By installing an input redirection tool in parallel with the SDE system, this problem can be minimized. Redirecting input between PCs connected to the shared SDE would allow each user to manipulate other users' windows.

To help differentiate windows owned by multiple users on a shared SDE, each user may select a different GUI theme on their PC. As the users move their GUI objects to the shared SDE, the themes are retained, offering a visual cue to indicate the owner of each object.

Because the SDE composites windows on the display, windows from multiple sources may overlap and intermix as though they all originated from the same source. This is shown in Figure 5.9. Currently, social protocol must be used to resolve conflicts between users sharing a display. For example, if user *A* places a window such that it entirely user *B*'s window, user *B* has no way to move the obscuring window to get at his or her window underneath. Again, an input redirection tool would mitigate this problem as it would allow user *B* to interact with user *A*'s windows.

A somewhat pedantic problem manifests itself if multiple users try to mirror their local displays

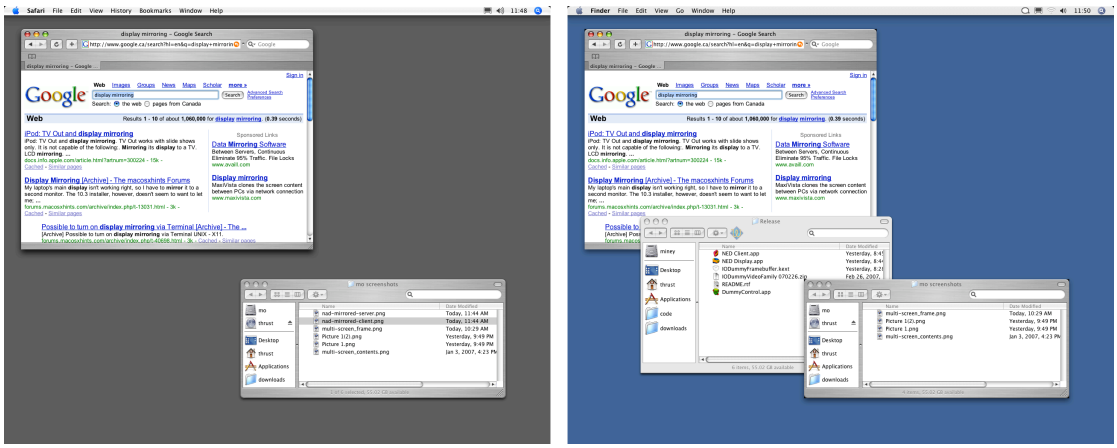

Figure 5.9: An SDE compositing multiple sources. A user's display (a) is mirrored on an SDE display (b). In addition, a second user has placed an additional window on the SDE display. Note how it has been inserted between the two windows from (a), overlapping one and being overlapped by the other.

to a shared SDE. In this situation, global GUI objects such as menu or task bars may collide due to their fixed position on-screen.

5.5 Empirically Determined Performance Results

SDE client performance was measured using the fastest PC on-hand, a 2.33 GHz Intel Core 2 Duo MacBook Pro with 3 GB of RAM. A 1.42 GHz PowerPC G4 Mac Mini with 1 GB of RAM coupled to an 18″ 1280 x 1024 LCD display was used as the SDE display device. Both systems were running Mac OS 10.4.10, and data was transferred over a 100 Mb/s network switch. All measurements were taken with `Activity Monitor.app` on the client PC, and all values are approximate, with an estimated accuracy of ±5 per cent. While the Mac Mini has substantially less power than the laptop, the computations required on the display device are minimal, and assumed to be negligible for the purpose of these measurements. Processing resources appeared to be the primary limiting factor; minor additional network traffic did not affect the performance of the SDE system.

- System connected and idling:

  Refresh Rate 0 Hz

  CPU use 1.5 per cent

  Bandwidth use 0 KB/s

This measurement indicates baseline performance for an idle system: no windows have been placed on the SDE, and the cursor is motionless. A refresh rate of 0 Hz is appropriate.

- Cursor movement:

  Refresh Rate 60 Hz

  CPU use 3.3 per cent

  Bandwidth use 16 KB/s

  Moving the cursor on an SDE is fast and responsive, as only cursor position information is being transmitted.

- Window dragging:

  Refresh Rate 8 Hz

  CPU use 30 per cent

  Bandwidth use 4 KB/s

  Dragging a 500 × 500 pixel window with static contents on the SDE consumes a relatively large amount of CPU time. This high CPU usage is due to the fact that the *block movement* window server callback does not work with the proxy framebuffer. Because of this, both window movement and window update events generate a screen changed callback, and the SDE Client software must compute the difference between current and previous window images to determine the correct update type. Once the SDE Client software has determined that a window move event has occurred, it only sends the updated position of the window, resulting in low network usage.

  The 8 Hz refresh rate is marginally lower than the minimum 10 Hz necessary to effectively simulate motion.

- Media playback:

  Refresh Rate 5 Hz

  CPU use 40 per cent

  Bandwidth use 1.5 MB/s

A 624 × 352 pixel movie was played in a 630 × 400 pixel window. Recall that because windows were not further subdivided into changed regions, the additional pixels used by the window borders must be sent on each refresh as well.

The 5 Hz frame rate is significantly lower than the minimum 10 Hz necessary to effectively simulate motion, and far lower than the native 24 Hz frame rate of the source video.

- Intermittent update:

  Refresh Rate 2 Hz

  CPU use 10 per cent

  Bandwidth use 35 KB/s

  If the window contents update intermittently, for example due to a blinking insertion point or sporadic typing, the SDE Client software only sends changed frames. This lowers network traffic significantly, at a cost of increased CPU usage. Even though the SDE Client software is notified by the window server when screen contents have changed, computing image differences to determine which window needs updating utilizes 10 per cent of the CPU for a 500 × 500 pixel window with an insertion point that blinks at 2 Hz. Once again this CPU utilization is due to not subdividing windows further, and requiring the whole 500 × 500 pixel region to be compared on each update event.

For the SDE to provide acceptable performance, it should be able to maintain a minimum refresh rate of 15 Hz for any of the operations listed above. Re-testing on more substantial hardware (i.e., a desktop PC) may yield such gains.

5.6 Requirements Re-visited

To determine if the SDE met the design set out in Chapter 4, the requirements from Section 5.1 will be reviewed.

- Connectivity:

  Requirement 1 *Require only one hardware port on the* PC *for an arbitrary number of satellite display devices.* **Satisfied in Section 5.2.2.**

Requirement 2 *Redesign existing link protocols to support satellite display devices with arbitrarily large or small device resolutions.* **Satisfied in Section 5.2.6.1.**

Requirement 3 *Include computational capabilities in the satellite display, including basic framebuffer memory and a* GPU *that supports minimal 2D graphics operations.* **Satisfied in Section 5.2.5.1.**

- Usability:

  Requirement 4 *Enable ease of device discovery and configuration using existing networking technologies such as DHCP and the ZeroConf service discovery protocol.* **Satisfied in Section 5.2.6.1.**

  Requirement 5 *Maintain visual fidelity by only advertising the native resolution of the satellite display.* **Satisfied in Section 5.2.5.2.**

  Requirement 6 *Maximize compatibility by ensuring that existing legacy software just works, and does not need to be recompiled or specially built to support the satellite display.* **Partially satisfied in Section 5.2.3.**

- Multiple users:

  Requirement 7 Allow multiple source PCs to simultaneously use a single satellite display by compositing the final output image on the satellite display device. **Satisfied in Section 5.2.2.**

  Requirement 8 Enable multi-user support by allowing each user to connect to a satellite display with their own PC. **Satisfied in Section 5.2.5.1.**

  Requirement 9 *Implement privacy measures by only allowing connected users to manipulate their own* GUI *objects, not those of other users.* **Satisfied by requirement 12.**

- Technical:

  Requirement 10 *Determine the feasibility of creating a prototype satellite display system given contemporary hardware and software, and development resources.* **Satisfied by overall implementation.**

  Requirement 11 *Do not allow display forwarding, i.e., a satellite display may not act as a client for a further satellite display.* **Not satisfied.**

Requirement 12 *Do not provide input redirection services between clients sharing a satellite display.* **Satisfied in Section 5.4.1.**

Of the twelve requirements specified at the beginning of this chapter, ten were met by the SDE, one was partially met, and one were not. The two requirements that were not fully met (maximize compatibility, and disallowing display forwarding) were dropped due to a lack of resources, as will be discussed next.

Providing a framebuffer compatible with all third-party application software requires the cooperation of the operating system and graphics card manufacturers; with their support, it would be straightforward to draw to an existing hardware framebuffer and copy the resulting bitmaps directly to the network interface. This would also provide a much-needed performance optimization.

Explicitly disallowing display forwarding was a feature that did not seem necessary in the end as this was a worst-case requirement. In practice this is unlikely to cause problems; a usage policy of not running the SDE Client and Server software simultaneously on one PC will suffice.

5.7 Contributions

The prototype software presented in this chapter provides a proof of concept technical implementation of a satellite display system. In the process, the SDE demonstrated that it is possible to build an *ad hoc*, easy to configure, compositing network display using current operating systems and PC hardware. It is hoped that by demonstrating that it is possible to overcome the considerable technical hurdles standing in the way, satellite displays will open the door to continued interest and further work on advancing the usability of multi-display environments. By providing complete source code for the current system (see Appendix D), this thesis provides a framework for future development.

The SDE project has provided basic multi-user display sharing, while continuing to support familiar single-user application software. It accomplishes this in a lightweight fashion, supporting *ad hoc* use of shared display space.

In the process of working through the technical aspects of this project, this thesis has documented previously undocumented features of Mac OS X. These features are often undocumented for good reason–they may be works in progress, deprecated at any time, or untested and unstable. However, cutting-edge research code is also a work in progress, and not intended to be used for

shipping products. By exposing and using some of the hidden features of an OS, it is hoped that the OS vendors will gain a deeper understanding of what third-party developers wish to do, and that they will help support research that moves the community forward as a whole.

5.8 Limitations

Setting aside the performance issues with the prototype implementation, the satellite display has some further limitations. As the goal was to faithfully reproduce a multi-display configuration, issues inherent to this setup are adopted by a satellite display. For example, in an extended desktop configuration, there may be long distances across many displays to traverse to reach the desired display. Due to the depth at which the SDE Client software gains access to the operating system, third party solutions to correct multi-display navigation issues ought to *just work*. For example, the MouseWarp utility introduced in Chapter 3 would allow a user to jump their cursor between both local and satellite displays. Other projects such as the Multi-monitor mouse [12] may also be adapted to this platform.

5.9 Future Work

There are two main technical areas where improvements would provide substantial benefit to the current SDE. First, working with operating system vendors may yield the system APIs necessary to provide proxy displays and efficiently determine which windows need to be captured. A significant engineering effort was spent on this aspect of the project, and the lack of information led to performance and feature limitations in the SDE. Second, a custom network protocol implementation would increase network performance by lowering both the bandwidth and computational overhead of the current image serialization scheme. This protocol may be similar to the existing RFB protocol used by VNC, with the necessary extensions to support independent video streams for each GUI object being transmitted. More importantly, such a custom protocol would allow interoperability between SDE Client and Server software running on different architectures. Achieving this would also open the door to a custom hardware implementation of a satellite display, which was an initial long-term goal of this thesis.

5.10 Summary

The prototype satellite display system, SDE, provides a framework that closely models a standard single computer multi-display setup. It does this by allowing the user of a client PC to drag windows and other GUI objects to a display that is not connected via standard DVI or VGA protocols. The SDE display device composites incoming graphics on the display itself, allowing it to be driven by multiple source PCs simultaneously. This allows multiple users to share and collaborate with their familiar single-user software

By attempting to maintain a faithful reproduction of a traditional multi-display system, it is hoped that the cost of entry to the satellite display system is minimized, leading to high rates of user adoption.

Chapter 6

Conclusion

After considering previous work (both academic and commercial) related to MDEs, two software components were developed for this thesis. The first was a utility that allows users to configure multi-display PCs in several ways, augmenting the standard mirrored and extended desktop modes. The second was a new display device architecture that allows multiple displays to be connected to a single physical port on a PC. In addition, multiple PCs may be connected to a single display, allowing multiple users to share the display simultaneously.

6.1 Contributions

The Reflect display configuration utility works well and attains its intended goal of expanding the number of available multi-display configurations. Reflect is augmented by MouseWarp, an auxiliary tool to enable further usage of the disjoint configuration mode that Reflect provides. Both tools have been released as an open source project that may be useful to more than just the research community. Several scenarios have been identified that take advantage of the additional multi-display configuration modes that the Reflect utility provides. As MDEs become more commonplace, additional configuration needs may be discovered. The flexibility of the Reflect utility should allow support for these new configurations.

In previous work I developed the Mighty Mouse project which allowed the input devices (e.g., keyboard and mouse) of any PC to control a networked, co-located PC [15]. The satellite display concept is a natural complement to that work. Rather than distributing input capabilities throughout an environment, the satellite display is a shared output device. As such, the satellite display concept fills a gap in a larger vision of co-located shared input and output.

The success that the Satellite Display Emulator (SDE) achieves is more reserved. As proof-of-concept software, it demonstrates that the necessary pieces are in place today to build a networked display. The `IOProxyVideoFamily`, a sub-project of the SDE, provides rare documentation and a sample implementation of this arcane corner of Mac OS X driver development.

6.2 Lessons Learned

The primary difficulty encountered while conducting the research reported in this thesis was the development of the SDE. While an ideal satellite display seemed straightforward on paper, implementing it was a daunting task. Throughout development, success always seemed just over the next hill, but progress was inevitably met with another technical mountain to ascend once each hill was surmounted. Looking back, it may have been beneficial to also build and test a lo-fi or *Wizard of Oz* prototype that explored the usability of the satellite display concept before implementing the SDE. Such a prototype would augment the feasibility findings made possible by the SDE by allowing us to focus on interaction techniques without having to rely on efficient and robust software.

6.3 Future Work

As software projects grow in complexity, it becomes increasingly difficult for a single developer to implement a complicated project with sufficient quality. The SDE is clearly prototype, proof of concept software: it establishes the possibility of a working system in this domain, but lacks the stability and features required to use it on a daily basis. A complete re-write, engineered from the start for cross-platform interoperability and efficient use of network and processor capacity, is probably necessary before further explorations should be made.

Once a stable implementation is achieved, the satellite display concept requires formal evaluation. Such an implementation needs to be benchmarked against both the performance of traditional multi-display configurations, and against other MDE management suites to determine the utility of its particular shared display paradigm. As mentioned in Section 6.2, the latter task may be accomplished in part by alternative prototyping methods.

Finally, the capabilities of shared display systems to support collaborative tasks needs further exploration. From the current vantage point, it is possible to imagine how a satellite display may be used within the context of the surrounding constraints: PCs, modal GUIs, and relatively slow networks. However, it will only be possible to fully realize the potential of this multi-display paradigm with the availability of a working implementation and future advances in multi-user operating systems. This is a chicken and egg process, and must be approached iteratively.

6.4 Recommendations

As demonstrated by this research, future exploration in the multi-display domain will be difficult unless operating system vendors provide improved access to the underlying functionality. In order to support third party developers, it would be necessary for Apple to fully expose the APIs for the novel technologies they develop. To successfully interoperate with Windows, Microsoft would need to do the same.

6.5 Conclusions

The wheel of reincarnation concept explained by Myer and Sutherland [52] continues to hold to this day. Van Dam *et al.* may not have foreseen the rapid miniaturization of CPUs, but their approach to push general purpose computing capabilities away from the central CPU towards a display device was effective and timely, leveraging the maximum amount of performance from existing hardware and paradigms [94]. Perhaps the next 30 years will bring a revolution in display technology that unbinds display devices from the PCs they have become tethered to. Until then, we must use what is currently available in the best way possible.

Bibliography

[1] Inc. Advanced Micro Devices. ATI Avivo. http://ati.amd.com/technology/Avivo/index.html, retrieved 2007-07-03.

[2] Apple Computer, Inc. Apple remote desktop. http://apple.com/remotedesktop/, retrieved 2006-05-31.

[3] Apple Computer, Inc. IOHIDSystem.cpp. http://www.opensource.apple.com/darwinsource/10.4.9.ppc/IOHIDFamily-185.19/IOHIDSystem/IOHIDSystem.cpp, retrieved 2007-07-03.

[4] Apple Computer, Inc. Leopard Technology Overview: Resolution Independence. http://developer.apple.com/leopard/overview/index.html, retrieved 2007-03-09.

[5] Apple Computer, Inc. Memory Management in Mac OS X. http://developer.apple.com/documentation/Performance/Conceptual/ManagingMemory/Articles/AboutMemory.html, retrieved 2006-06-21.

[6] Apple Computer, Inc. Technical Q&A QA1197: Mapping kernel memory to user space on Mac OS X 10.2. http://developer.apple.com/qa/qa2001/qa1197.html, retrieved 2006-09-24.

[7] Ritchie Argue, Kellogg S. Booth, and Kori M. Inkpen. Reflect & satellite displays: advanced multi-display configuration. Poster in Extended Abstracts, Graphics Interface, 2007.

[8] Ronald Baecker. Digital video display systems and dynamic graphics. In *SIGGRAPH '79: Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, pages 48–56, New York, NY, USA, 1979. ACM Press.

[9] Barco. Barco. http://barco.com/, retrieved 2007-07-03.

[10] Bartels Media. Maxivista. http://maxivista.com/, retrieved 2006-05-12.

[11] Patrick Baudisch, Edward Cutrell, Ken Hinckley, and Robert Gruen. Mouse ether: accelerating the acquisition of targets across multi-monitor displays. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1379–1382, New York, NY, USA, 2004. ACM Press.

[12] Hrvoje Benko and Steven Feiner. Multi-monitor mouse. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1208–1211, New York, NY, USA, 2005. ACM Press.

[13] Lior Berry, Lyn Bartram, and Kellogg S. Booth. Role-based control of shared application views. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 23–32, New York, NY, USA, 2005. ACM Press.

[14] Jacob T. Biehl and Brian P. Bailey. Aris: an interface for application relocation in an interactive space. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 107–116, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[15] Kellogg S. Booth, Brian D. Fisher, Chi Jui Raymond Lin, and Ritchie Argue. The "Mighty Mouse" multi-screen collaboration tool. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 209–212, New York, NY, USA, 2002. ACM Press.

[16] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for gpus: stream computing on graphics hardware. *ACM Trans. Graph.*, 23(3):777–786, 2004.

[17] David Chaiken. x2x.
`http://x2x.dottedmag.net/`, retrieved 2006-03-02.

[18] Matthew Chapman. rdesktop.
`http://www.rdesktop.org`, retrieved 2006-05-19.

[19] Olivier Chapuis and Nicolas Roussel. Metisse is not a 3d desktop! In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 13–22, New York, NY, USA, 2005. ACM Press.

[20] Stuart Cheshire. Zeroconf.
`http://www.zeroconf.org/`, retrieved 2007-01-15.

[21] Codeweavers, Inc. CrossOver Mac.
`http://codeweavers.com/products/cxmac/`, retrieved 2007-01-22.

[22] DDWG Promoters. Digital Visual Interface DVI, Revision 1.0.
`http://www.ddwg.org/lib/dvi_10.pdf`, retrieved 2007-03-09.

[23] Digital Content Protection LLC. High-bandwidth Digital Content Protection System, Revision 1.3.
`http://www.digital-cp.com/home/HDCP_Specification%20Rev1_3.pdf`, retrieved 2007-07-03.

[24] Element Labs, Inc. Element labs, inc.
`http://elementlabs.com/`, retrieved 2007-07-03.

[25] Element Labs, Inc. Element labs projects.
http://www.elementlabs.com/category/projects/, retrieved 2007-07-03.

[26] John Elliot. Dual-Head operation on vintage PCs.
http://www.seasip.info/VintagePC/dualhead.html, retrieved 2006-05-15.

[27] Rik Faith, Kevin E. Martin, and Randall Frank. Distributed Multihead X.
http://dmx.sourceforge.net/, retrieved 2006-05-12.

[28] Jonathan Grudin. Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 458–465, New York, NY, USA, 2001. ACM Press.

[29] Jonathan Grudin. Return on investment and organizational adoption. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 324–327, New York, NY, USA, 2004. ACM Press.

[30] Vicki Ha, Kori Inkpen, Jim Wallace, and Ryder Ziola. Swordfish: user tailored workspaces in multi-display environments. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1487–1492, New York, NY, USA, 2006. ACM Press.

[31] Chris Hanson. Xydra.
http://chanson.livejournal.com/15605.html, retrieved 2006-06-21.

[32] Havok.com Inc. Havok FX.
http://www.havok.com/content/view/187/77/, retrieved 2007-02-08.

[33] HDMI Licensing, LLC. High Definition Multimedia Interface Specification Version 1.3a.
http://www.hdmi.org/download/HDMISpecification13a.pdf, retrieved 2007-03-09.

[34] D. Austin Henderson, Jr. and Stuart Card. Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Trans. Graph.*, 5(3):211–243, 1986.

[35] Frederik Hubinette. x2vnc.
http://fredrik.hubbe.net/x2vnc.html, retrieved 2006-03-02.

[36] Greg Humphreys and Pat Hanrahan. A distributed graphics system for large tiled displays. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 215–223, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[37] Dugald Ralph Hutchings, Greg Smith, Brian Meyers, Mary Czerwinski, and George Robertson. Display space usage and window management operation comparisons between single monitor and multiple monitor users. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 32–39, New York, NY, USA, 2004. ACM Press.

[38] Heather M. Hutchings and Jeffrey S. Pierce. Understanding the whethers, hows, and whys of divisible interfaces. In *AVI '06: Proceedings of the working conference on Advanced visual interfaces*, pages 274–277, New York, NY, USA, 2006. ACM Press.

[39] Peter Hutterer and Bruce H. Thomas. Groupware support in the windowing system. In *AUIC '07: Proceedings of the eight Australasian conference on User interface*, pages 39–46, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[40] Shahram Izadi, Harry Brignull, Tom Rodden, Yvonne Rogers, and Mia Underwood. Dynamo: a public interactive surface supporting the cooperative sharing and exchange of media. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 159–168, New York, NY, USA, 2003. ACM Press.

[41] Brad Johanson, Greg Hutchins, Terry Winograd, and Maureen Stone. Pointright: experience with flexible input redirection in interactive workspaces. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 227–234, New York, NY, USA, 2002. ACM Press.

[42] Heather Lanigan, Jay Cotton, and Paul Anderson. Xinerama. `http://sourceforge.net/projects/xinerama/`, retrieved 2006-05-12.

[43] Tom LaStrange. swm: An X Window Manager Shell. In *USENIX Conference Proceedings*, 1990.

[44] M. Macedonia. The gpu enters computing's mainstream. *Computer*, 36(10):106–108, 2003.

[45] I. Scott MacKenzie and William Buxton. Extending fitts' law to two-dimensional tasks. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 219–226, New York, NY, USA, 1992. ACM Press.

[46] Matrox Graphics Inc. Multi-display leadership. `http://www.matrox.com/graphics/en/corpo/about/multidisplay/home.php`, retrieved 2007-02-08.

[47] Matrox Graphics Inc. Graphics eXpansion Modules. `http://matrox.com/graphics/offhome/gxm.cfm`, retrieved 2006-05-31, 2006.

[48] Microsoft Corporation. Microsoft Virtual PC. `http://microsoft.com/windows/virtualpc/default.mspx`, retrieved 2007-01-22.

[49] Microsoft Corporation. Remote desktop connection. `http://microsoft.com/windowsxp/downloads/tools/rdclientdl.mspx`, retrieved 2006-05-15.

[50] Microsoft Corporation. Windows SideShow. http://www.microsoft.com/windows/products/windowsvista/features/details/sideshow.mspx, retrieved 2007-07-03.

[51] G. E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

[52] T. H. Myer and I. E. Sutherland. On the design of display processors. *Commun. ACM*, 11(6):410–414, 1968.

[53] Brad A. Myers. The pebbles project: using PCs and hand-held computers together. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, pages 14–15, New York, NY, USA, 2000. ACM Press.

[54] Miguel A. Nacenta, Samer Sallam, Bernard Champoux, Sriram Subramanian, and Carl Gutwin. Perspective cursor: perspective-based interaction for multi-display environments. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 289–298, New York, NY, USA, 2006. ACM Press.

[55] Robert Nation. Fvwm window manager. http://fvwm.org, retrieved 2007-01-22.

[56] National Semiconductor. FPD-Link PCB and Interconnect Design-In Guidelines. http://www.national.com/an/AN/AN-1085.pdf, retrieved 2007-03-09.

[57] National Semiconductor. Open LVDS Display Interface (OpenLDI) Specification, v0.95. http://www.national.com/appinfo/lvds/openldi.pdf, retrieved 2007-03-09.

[58] Neslo Software, Inc. Desktop rover. http://neslosoftware.com/ retrieved 2007-01-15.

[59] Netopia. Timbuktu pro remote control software. http://www.netopia.com/software/products/tb2/, retrieved 2006-03-02.

[60] NVIDIA Corporation. NVIDIA PureVideo. http://www.nvidia.com/page/purevideo.html, retrieved 2007-07-03.

[61] Mike Paquette. Re: Cgconfiguredisplayorigin. http://www.cocoabuilder.com/archive/message/cocoa/2005/4/13/133056, retrieved 2006-06-13.

[62] Mike Paquette. Why Apple didn't use X for the window system. http://developers.slashdot.org/comments.pl?sid=75257&cid=6734612, retrieved 2006-05-15.

[63] Parallels, Inc. Parallels desktop. http://parallels.com, retrieved 2007-01-22.

[64] pcworld.com. Samsung 940UX.
http://blogs.pcworld.com/digitalworld/archives/2007/02/samsung_goodbye.html, retrieved 2007-03-05.

[65] Havoc Pennington. Metacity.
http://ftp.acc.umu.se/pub/gnome/sources/metacity/, retrieved 2007-02-08.

[66] Shankar R. Ponnekanti, Brad Johanson, Emre Kiciman, and Armando Fox. Portability, extensibility and robustness in iros. In *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, page 11, Washington, DC, USA, 2003. IEEE Computer Society.

[67] Arash Ramin. Multiple monitor guide.
http://www.digitalroom.net/techpub/multimon.html, retrieved 2006-05-15.

[68] Ramesh Raskar, Michael S. Brown, Ruigang Yang, Wei-Chao Chen, Greg Welch, Herman Towles, Brent Seales, and Henry Fuchs. Multi-projector displays using camera-based registration. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 161–168, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[69] Realtime Soft. Ultramon.
http://www.realtimesoft.com/ultramon/, retrieved 2006-05-25.

[70] RealVNC Ltd. VNC.
http://realvnc.com, retrieved 2007-01-22.

[71] Ryan Rempel. Virtual framebuffer.
http://lists.apple.com/archives/darwin-drivers/2005/Aug/msg00039.html, retrieved 2006-09-24.

[72] Ryan Rempel. XPostFacto.
http://eshop.macsales.com/OSXCenter/XPostFacto/, retrieved 2006-05-31.

[73] David Reveman, Matthias Hopf, Dave Arlie, Adam Jackson, and Jon Smirl. Compiz.
http://www.go-compiz.org/, retrieved 2007-02-08.

[74] Tristan Richardson. The RFB protocol.
http://www.realvnc.com/docs/rfbproto.pdf, retrieved 2006-05-15.

[75] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[76] Julien Robert. Teleport.
http://abyssoft.com/software/teleport/, retrieved 2007-02-08.

[77] G. Robertson, M. Czerwinski, P. Baudisch, B. Meyers, D. Robbins, G. Smith, and D. Tan. The large-display user experience. *Computer Graphics and Applications, IEEE*, 25(4):44–51, 2005.

[78] Malcolm E. Rodgers, Regan L. Mandryk, and Kori M. Inkpen. Smart sticky widgets: Pseudo-haptic enhancements for multi-monitor displays. In Andreas Butz, Brian Fisher, Antonio Krüger, and Patrick Olivier, editors, *Smart Graphics*, Lecture Notes in Computer Science, pages 194–205. Springer, 2006.

[79] Manuel Roman and Roy H. Campbell. Gaia: enabling active spaces. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 229–234, New York, NY, USA, 2000. ACM Press.

[80] Samsung. Networkable large format tft displays for digital signage and point of information applications.
`http://www.samsung.com/uk/business/b2b/products/displays/public_displays/sm400pxn_and_sm460pn.htm`, retrieved 2007-07-03.

[81] Glen Sanford. Macintosh II.
`http://www.apple-history.com/?model=II`, retrieved 2006-05-15.

[82] Greg Schechter. Under the hood of the desktop window manager.
`http://blogs.msdn.com/greg_schechter/archive/2006/03/05/544314.aspx`, retrieved 2006-06-06.

[83] Robert W. Scheifler and Jim Gettys. The X window system. *ACM Trans. Graph.*, 5(2):79–109, 1986.

[84] Chris Schoeneman. Synergy.
`http://synergy2.sourceforge.net/`, retrieved 2006-03-02.

[85] Jason Seng. darwin-kernel mailing list.
`http://lists.apple.com/archives/darwin-kernel/2005/Apr/msg00074.html`, retrieved 2006-09-24.

[86] Stardock Corp. Multiplicity.
`http://www.stardock.com/products/multiplicity/`, retrieved 2007-01-17.

[87] Patrick Stein. ScreenRecycler.
`http://www.screenrecycler.com/`, retrieved 2007-01-17.

[88] R. Stevens, M. E. Papka, M. Kilgard, G. Humphreys, and T. Funkhouser. Commodity graphics accelerators for scientific visualization. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 527–529, Washington, DC, USA, 2001. IEEE Computer Society.

[89] Jason Stewart, Benjamin B. Bederson, and Allison Druin. Single display groupware: a model for co-present collaboration. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 286–293, New York, NY, USA, 1999. ACM Press.

[90] Desney S. Tan, Brian Meyers, and Mary Czerwinski. Wincuts: manipulating arbitrary window regions for more effective use of screen space. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1525–1528, New York, NY, USA, 2004. ACM Press.

[91] Vasily Tarasov. ZoneScreen. `http://www.zoneos.com/zonescreen.htm`, retrieved 2007-01-22.

[92] Mark Tyrrell. Wraparound. `http://www.digicowsoftware.com/detail?_app=Wraparound`, retrieved 2007-01-17.

[93] UDI Promoters. Unified Display Interface (UDI) Specification, Revision 1.0a Final. `http://www.udiwg.org/udi_spec/UDI_1_0.pdf`, retrieved 2007-03-09.

[94] A. van Dam, G. M. Stabler, and R. J. Harrington. Intelligent satellites for interactive graphics. *Proceedings of the IEEE*, 62(4):483–492, 1974.

[95] Video Electronics Standards Association. DisplayPort Standard. `http://www.vesa.org/Standards/free.htm`, retrieved 2007-03-09.

[96] Video Electronics Standards Association. VESA EDID Implementation Guide, Version 1.0. `http://www.vesa.org/Public/EEDIDguideV1.pdf`, retrieved 2007-03-09.

[97] VMware, Inc. VMware. `http://vmware.com`, retrieved 2007-01-22.

[98] Rudi De Vos and UltraSam. UltraVNC. `http://uvnc.com`.

[99] Ruigang Yang, David Gotz, Justin Hensley, Herman Towles, and Michael S. Brown. Pixelflex: a reconfigurable multi-projector display system. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 167–174, Washington, DC, USA, 2001. IEEE Computer Society.

Appendix A

IOProxyVideoFamily

A.1 Introduction

At the most basic level, a display device is represented on a computer by a **framebuffer**, or a contiguous region in memory which stores a raster representation of the current image being displayed. This memory is known as VRAM and is usually provided by the graphics hardware, although computers with integrated graphics processors may reserve a portion of the main system memory for use as the framebuffer. By mapping addresses in the framebuffer to locations, or pixels, on the display screen, a raster image may be represented in memory by setting each location in the framebuffer to an appropriate value. Once the operating system has written a full image into the framebuffer, supporting video hardware reads the framebuffer pixel by pixel, converting it into an electrical signal suitable to drive the attached hardware display device.

Newer video graphics hardware typically provide extended commands to enable 3D rendering or accelerated graphics operations such as region copies or moves. These commands may operate on the framebuffer in an optimized way, for example by allowing the graphics hardware to access memory in parallel, but ultimately produce the same net effect as the CPU manipulating the framebuffer directly; a new image is produced on the framebuffer for display by the output electronics. Since the level of accelerated support provided by various hardware manufacturers varies and acceleration is not guaranteed to exist, operating systems generally support drawing to a raw framebuffer as a fallback rendering technique.

The IOProxyVideoFamily suite of kernel extensions provides a combination of the above techniques to simulate a real framebuffer to the operating system. As far as the operating system knows, real hardware-supported VRAM is being made available to it. The key difference between a proxy framebuffer and a real framebuffer is the method for reading data out of it for eventual display to the user. While a real framebuffer is read once per screen refresh by dedicated hardware, the proxy framebuffer is never read by hardware; instead it may be read by operating system calls that copy portions of the framebuffer back into user-space accessible regions of memory for further

processing.

A.1.1 Mac OS X driver architecture

To discuss the design and implementation of the IOProxyVideoFamily, some background of the Mac OS X driver architecture, IOKit, is necessary.

In Mac OS X, device drivers are loaded and unloaded dynamically as the underlying hardware state changes (devices come online or are removed). This model provides flexibility for hot-plugged hardware such as PCMCIA cards and USB devices. In addition, a layered model is used to represent hardware from the base level of the system, up through various busses to a particular device. Each driver in a given layer publishes **nubs** which specify properties of the device. Drivers in the layer above may match on a nub in a lower level driver if the properties indicate that the nub provides services applicable for that driver. At the interface between two drivers, the driver publishing a nub is a **provider**, while the driver which matches on the nub is a **client**. If a client doesn't match on a nub, it is discarded and the next driver is tried. Each driver that successfully matches a nub is given a probe score–a measure of the fitness of that client to interact with the device. In the event of multiple matches, the client with the highest probe score assumes control of the device.

Typically a driver will publish a nub for each independent device capability, for example a PCI bus driver would have a nub for each PCI slot in the system, and a driver for a multi-protocol communications card may publish a nub for each protocol it supports over the physical medium. Drivers may publish zero or more nubs for their device, but only one client may match per nub.

Similarly, clients may possess multiple **personalities**, or matching interfaces for nubs. To illustrate the need for multiple personalities, consider a manufacturer that has several models of the same type of device, each slightly differing in capability. By writing a single driver with multiple personalities, it may be used for all the devices even though they advertise nubs from the layer below.

At a leaf node in the device tree, drivers no longer publish nubs for further in-kernel drivers to match on but instead provide **user clients** for user space applications to access.

A.1.2 Virtual devices

Because there is no real hardware to back virtual devices, IOKit provides IOResources, a software only nub for drivers to connect to. IOResources publishes a single nub, but unlike regular providers,

permits multiple personalities to match on it to allow multiple virtual device drivers to exist simultaneously.

A.1.3 IOFramebuffer

Framebuffers on Mac OS X are leaf node drivers in the IOKit, and are subclasses of the IOFramebuffer abstract base class. As Figure A.1 illustrates, instances of IOFramebuffer have one user client, the Window Server. In turn, they are typically clients of proprietary hardware drivers that are responsible for managing the graphics hardware, including VRAM. During initialization, the IOFramebuffer portion of the driver queries its provider for a memory range to use as the framebuffer.

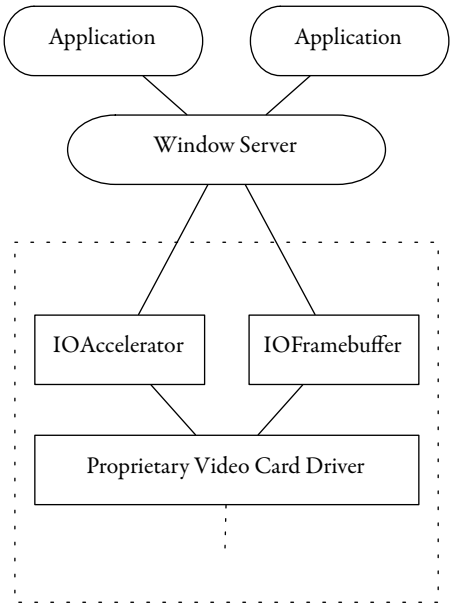

Figure A.1: IOFramebuffer Components

A.1.4 Related work

As a simulated framebuffer offers little functionality without additional supporting software, they are difficult to find information on.

Although the source to the IOKit IOFramebuffer abstract base class is available, it is rather opaque and offers little information to clarify what is necessary to write a custom subclass. An old IOKit sample project, AppleSampleFramebuffer, may have been of use, but it was difficult to find documentation for it. Much of the other work in this area is comprised of closed source drivers

written by graphics card manufacturers who work directly with the OS vendors, and as such, useful public documentation is difficult to find.

Searching outside of the Apple sample code archives turned up Xydra, an early attempt at a remote display [31]. Xydra includes a limited virtual IOFramebuffer implementation, but no source or instructions are available, and the project crashes immediately on MacOS 10.4.

Finally, Ryan Rempel wrote XPostFacto, an open-source project that allows Mac OS X to run on old, no longer supported Apple hardware [72]. Part of this project, OpenControlFramebuffer, is an IOFramebuffer subclass that illustrates the required functions and their rough implementation. Rempel has also provided a basic virtual framebuffer implementation, with the useful instruction that the driver must be loaded at boot rather than run time [71].

A.2 Implementation

The initial virtual framebuffer prototype consisted of a single IOProxyFramebuffer instance that matched on IOResources. This method is adequate for providing a single framebuffer, however simulating multiple heads would require additional personalities to match against IOResources. Further, the Window Server has exclusive access to an IOFramebufferUserClient, meaning that it is not possible for other user space applications to communicate with IOFramebuffer instances [85]. This hampers the ability to toggle virtual heads on and off, as well as being able to set custom resolutions at runtime.

To overcome the above issues, the project was split into two components. First, an IOProxyVideoCard was created to simulate a multi-headed graphics card. This serves multiple purposes: it publishes multiple nubs for IOProxyFramebuffer to match on, provides a user client that can receive communication from outside the kernel, and manages the memory that simulates VRAM. Second, a stripped down IOProxyFramebuffer matches on the nubs published by the new IOProxyVideoCard, loads screen resolution configuration information from a nub, and exposes a framebuffer aperture to the Window Server. This architecture is illustrated in Figure A.2.

A.2.1 IOProxyVideoCard

To best support the satellite display project, an IOProxyVideoCard should be able to support a dynamic configuration of an arbitrary number of heads, each of arbitrary size, matching remote extended desktops as they appear and disappear on the network.

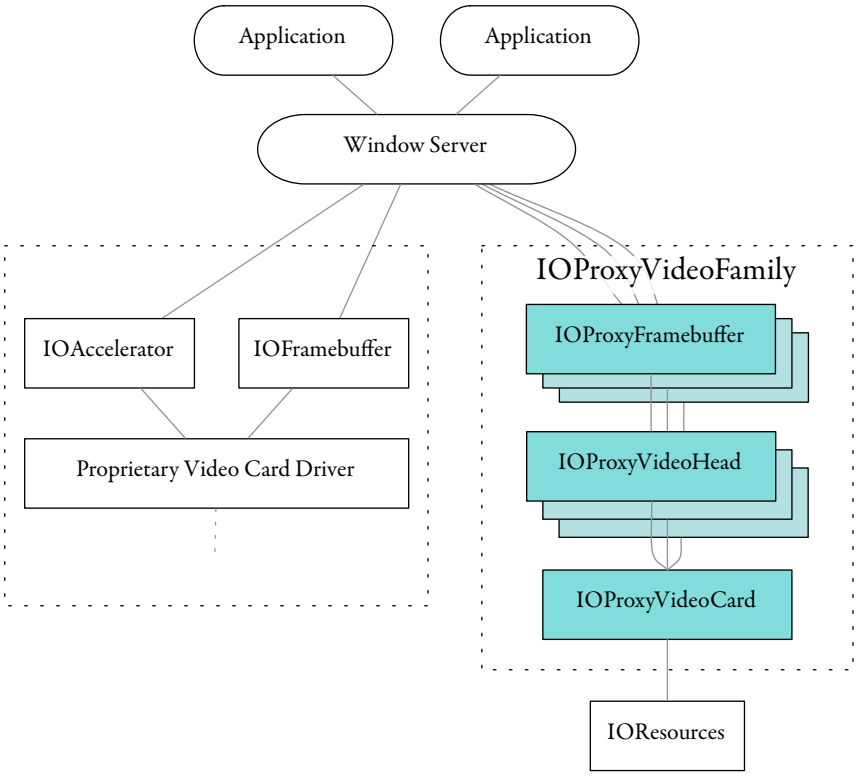

Figure A.2: IOProxyVideoFamily Components

Due to restrictions imposed by the IOKit architecture, the engineering effort required to realize this goal, or determine its feasibility, is beyond the scope of this project. First, the total amount of VRAM used by each IOFramebuffer subclass must be set at boot time, and may not be modified while the system is running, making it impossible to dynamically create arbitrarily large framebuffers at runtime. Secondly, there is no documented way to initiate a hardware interrupt from within a virtual device driver, which means that events such as display addition and removal are also not possible at runtime. As such, the current IOProxyVideoCard is restricted to a static number of heads, each with a fixed maximum resolution.

A.2.1.1 Static configuration

Each nub of the IOProxyVideoCard represents a virtual display connection, or head. The necessary information for configuring a simulated graphics card is expressed by a list of heads, each with a maximum allowable resolution as well as a list of valid smaller resolutions. The XML representation of a graphics card with 3 heads is illustrated in Figure A.3. A maximum resolution for each head

is required due to memory management constraints of the Window Server, as explained in the following section.

```
<key>com_doequalsglory_driver_HeadList</key>
<array>
    <!-- each dictionary in this array represents one virtual head -->
    <dict>
        <!-- this dictionary has explicit parameters set -->
        <key>enabled</key>
        <true/>
        <key>height</key>
        <integer>1050</integer>
        <key>maxHeight</key>
        <integer>1200</integer>
        <key>maxWidth</key>
        <integer>1920</integer>
        <key>name</key>
        <string>DEG,ProxyVideoHead_A</string>
        <key>width</key>
        <integer>1400</integer>
    </dict>
    <!-- these two dictionaries are empty - they create two
         virtual heads with default settings -->
    <dict/>
    <dict/>
</array>
```

Figure A.3: IOProxyVideocard static configuration options

A.2.1.2 Memory allocation

Due to performance requirements of the Window Server, and the reasonable assumption for real hardware that the VRAM size doesn't change during the operation of a computer, this aspect of the hardware must be simulated faithfully. Specifically this means that shortcuts cannot be taken, such as dynamically allocating memory for the framebuffers in use at any given time. It also imposes the constraint that the driver must load before the Window Server does, and unload after it dies. As mentioned in the previous paragraph, each head has a maximum allowable resolution specified in the configuration file, and these values are parsed when the IOProxyVideoCard is loaded. Using these values, enough memory is allocated at load time to represent the set of the framebuffers specified.

Prototype versions of the IOProxyFramebuffer used the `IOBufferMemoryDescriptor::withCapacity()` function call to allocate simulated VRAM in the kernel. This call could allocate a limited amount of memory, less than the amount required to support two 1920 by 1200 pixel framebuffers (approximately 18MB). The Apple Technical Q&A QA1197 explains why this is the case, and suggests a replacement function that is capable of allocating much more memory from outside the kernel [6].

A.2.1.3 Dynamic configuration

Unlike the IOProxyFramebuffer, there is no restriction on which user space processes may talk to the user client of an IOProxyVideoCard. With this approach, dynamically configuring the IOProxyFramebuffer should be possible as the IOProxyVideoCard can make changes and trigger a refresh of the display system. There are three target settings for this configuration step: the connection state (connected, disconnected), a custom resolution, and the display name.

A configuration utility was created to interface with the IOProxyVideoCard, updating its parameters from user space. Once the configuration is stored in the IOProxyVideoCard, it must notify the appropriate IOProxyFramebuffer instances that parameters have changed. In a real hardware driver, the IOProxyFramebuffer base class responds to an interrupt generated by the underlying hardware, and re-reads applicable configuration information. Initial research indicates that it is not possible for a provider to simulate or initiate hardware interrupts to a client within the IOKit architecture, which has hampered development in this area. Further exploration of this limitation is beyond the scope of this thesis.

A.2.2 IOProxyFramebuffer

Until a method is devised to synthesize hardware interrupts in the IOProxyVideoCard, runtime configuration of the IOProxyFramebuffer is limited. To simulate the ability to connect and disconnect a virtual display at runtime, a resolution of 0 × 0 pixels is provided by the IOProxyFramebuffer. The user can then enable the framebuffer by selecting a valid resolution, or disconnect by selecting 0 × 0.

A.3 Usage

The IOProxyVideoFamily suite of kernel extensions, IOProxyFramebuffer and IOProxyVideoCard, are installed by copying them into the `/System/Library/Extensions` folder, and invalidating the kernel extension cache, as shown in Figure A.4. Note that unlike other kernel extensions, IOFramebuffer subclasses must be instantiated at boot time, and will not function if they are loaded via `kextload` at runtime.

```
sudo cp -R IOProxyFramebuffer.kext /System/Library/Extensions/
sudo cp -R IOProxyVideoCard.kext /System/Library/Extensions/
sudo touch /System/Library/Extensions
sync
reboot
```

Figure A.4: IOProxyVideoFamily Installation Procedure

After rebooting, the additional virtual displays provided by IOProxyVideoFamily are made available to the system, as shown in Figure A.5. In this example, the display with the menu bar is a real hardware display, while the remaining three are virtual. This screenshot was taken on a Mac Mini, which has a single video output and no graphics hardware expansion capabilities. As far as the operating system or 3rd party applications know, they are real, hardware framebuffers. These can now be used with the satellite display system, or other remote display application.

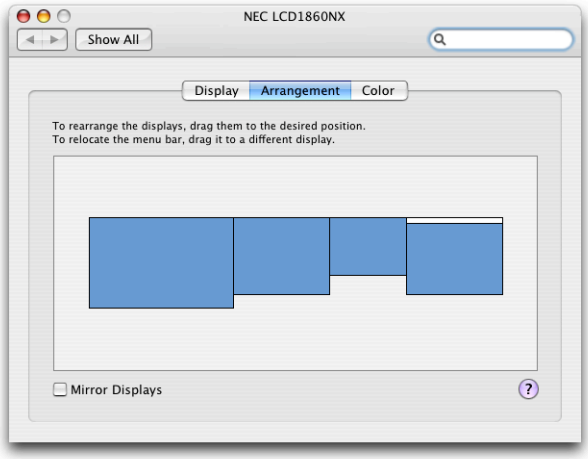

Figure A.5: Three virtual displays

A.4 Limitations and Future Work

The IOKit provides extensive capabilities for configuring and managing drivers. While the requirements of the ideal IOProxyVideoFamily appear to be attainable within the IOKit framework, a more significant engineering effort would be required to do so. As this piece is simply a shim to work around other limitations in the Window Server, applying the necessary engineering effort to create this ideal configuration is beyond the scope of this thesis.

With unlimited resources, the following features are candidates for further development towards more accurately simulating real graphics hardware:

1. Add and remove nubs at runtime, or extend the simulation one level deeper and create a virtual graphics card bus with hot-swap graphics card capabilities. Nub addition and removal may be simulated by toggling nubs on and off. It is not clear whether current versions of Mac OS X can hot-swap real graphics cards; initial research suggests that this is not possible.

2. Synthesize and initiate hardware interrupts from within a virtual device driver, to allow the IOProxyVideoCard to trigger changes in a framebuffer, such as resolution, bit depth, or the display name.

3. Falsify Extended Display Identification Data (EDID) information to provide more descriptive remote display names to the user.

4. Subclass IOAccelerator to support some 2D acceleration operations, such as block moves.

Finally, when considered in the context of the satellite display, memory is not used efficiently by the proxy framebuffer. The IOProxyFramebuffer consumes enough memory to represent a full framebuffer, and thus its overall size and depth is only limited by the amount of available memory. In addition, the window server keeps its own offscreen copy of each window, and only composites down to the framebuffer as a last step in the rendering pipeline. The private window grab methods on Mac OS X allow windows to be captured directly from the window server, and so the windows never need to be composited out to the IOProxyFramebuffer in order to be captured. As a result, a more efficient implementation would be built directly into the window server, providing a logical region in which to place windows, but not backing this with a real framebuffer.

A.5 Conclusion

The IOProxyVideoFamily is a software-only implementation of multi-headed graphics hardware, intended to provide a simulated display for the satellite display project. It succeeds in this capacity, however development difficulties prevented it from being as robust and flexible as initially planned. Because the IOProxyFramebuffer was always considered a workaround for a larger problem, further development to overcome the issues faced by this piece were deemed an unproductive use of resources. For full source code to the IOProxyVideoFamily project, see Appendix D.

Appendix B

Code spelunking

To a 3rd party developer on a closed- (or predominantly closed-) source operating system, the implementation of system calls is often opaque: the results are clear, but the method by which they are accomplished are not. With a good system design this usually doesn't matter; it's the results that the programmer is after, and it doesn't matter how they are generated. Occasionally it is necessary to delve into the workings of a system call; for example, when a bug is suspected, or when the calls are not provided at a fine enough granularity to allow the functionality desired.

The development of the software used in this thesis skirts the edges of what is possible on current operating systems. As system implementers can't know ahead of time all potential applications 3rd party developers may develop, APIs provided are occasionally inadequate. Even if all the required functionality is in place to achieve an obscure result, its use may not be sufficiently well understood by those outside the teams that created it. As a result, it was often necessary to explore the operating system to gain a greater understanding of it's inner workings. Further, on a few occasions the existing operating system code implementing certain API calls needed to be modified to provide the necessary functionality.

Several freely available exploration tools will be discussed in this Appendix. Patching the CoreGraphics framework to allow for arbitrary display placement will be used to illustrate the exploration and modification process.

B.1 Locating System Calls

CoreGraphics is a system **framework** that provides a means for 3rd party developers to configure extended desktop display placement via the `CGConfigureDisplayOrigin()` function call. In order to allow for arbitrary display placement as desired for the Reflect project discussed in Chapter 3, this call must be patched so that it does not reposition the display to a valid location when given invalid parameters.

Before starting the search, `CGConfigureDisplayOrigin()` was tested with invalid parameters, positioning a display such that it partially overlapped or was separated from another display. In both cases, `CGConfigureDisplayOrigin()` succeeds, but in the process moves the display to the nearest location that ensures that the displays do not overlap or have edges disjoint from other displays. From this it is deduced that either `CGConfigureDisplayOrigin()` is moving the display itself, or is calling a helper function to do so.

The function `CGConfigureDisplayOrigin()` is provided by the CoreGraphics framework. The executable binary for this framework is found at:
`/System/Library/Frameworks/ApplicationServices.framework/Frameworks/CoreGraphics.framework/CoreGraphics`.

A selection of open-source code exploration utilities are available with Apple's free Developer Tools. Two tools that operate on object files are used here: `nm` lists the symbol table, and `otool` disassembles and displays the corresponding assembly source code.

Using `nm` to dump symbols from the `CoreGraphics` binary, as shown in Figure B.1, indicates that `CGConfigureDisplayOrigin()` does indeed exist in that file, along with a function named `CGSConfigureDisplayOrigin()`. The *T* in the column preceding the name indicates that the symbol exists in the Text, or executable code, section of the file. This indicates that it is a method name, unlike the `kCGSDisplayOrigin*` symbols, which are in the Data section and thus are global constants. An upper-case *T* is used to indicate that this symbol is exported, or available for use from outside the file it exists in.

```
> nm -g /System/Library/Frameworks/ApplicationServices.framework/
Frameworks/CoreGraphics.framework/CoreGraphics |grep DisplayOrigin

906ce404 T _CGConfigureDisplayOrigin
906c0484 T _CGSConfigureDisplayOrigin
a0374448 D _kCGSDisplayOriginX
a037444c D _kCGSDisplayOriginY
>
```

Figure B.1: Selected public symbols in the CoreGraphics framework

The listing in Figure B.2 indicates that the last opcode of `CGConfigureDisplayOrigin()` is a jump to `CGSConfigureDisplayOrigin()`. The `CGS` prefix in this context represents CoreGraphicsServer, or the window server component of the window manager.

```
> otool -tV /System/Library/Frameworks/
ApplicationServices.framework/Frameworks/
CoreGraphics.framework/CoreGraphics |grep -A 20
CGConfigureDisplayOrigin

_CGConfigureDisplayOrigin:
906ce404        or.     r4,r4,r4
906ce408        mfspr   r0,lr
906ce40c        stmw    r27,0xffec(r1)
906ce410        stw     r0,0x8(r1)
906ce414        or      r30,r3,r3
906ce418        stwu    r1,0xffa0(r1)
906ce41c        or      r29,r5,r5
906ce420        or      r28,r6,r6
906ce424        bne+    0x906ce430
906ce428        bl      _CGMainDisplayID
906ce42c        or      r4,r3,r3
906ce430        lwz     r0,0x68(r1)
906ce434        addi    r1,r1,0x60
906ce438        or      r3,r30,r30
906ce43c        or      r5,r29,r29
906ce440        mtspr   lr,r0
906ce444        or      r6,r28,r28
906ce448        lmw     r27,0xffec(r1)
906ce44c        b       _CGSConfigureDisplayOrigin
_CGConfigureDisplayMode:
>
```

Figure B.2: Assembly dump of `CGConfigureDisplayOrigin()`

Tracing through additional symbols and assembly in CoreGraphics with `otool` reveals the following set of calls made by `CGConfigureDisplayOrigin()`: `CGXCompleteDisplayConfiguration()`, `CGXLCorrectDisplays`, and `CGXLFindBestPositionForDisplay()`. The `nm` utility indicates that these are local symbols, and may not be called directly by code external to the CoreGraphics binary. `CGXLFindBestPositionForDisplay()` does what its name suggests, moving an improperly positioned display to the best position for it.

B.2 Patching

There are two main steps to the patching process: determining what to replace the existing code with, and implementing the binary patch.

B.2.1 Replacement code

A replacement function for `CGXLFindBestPositionForDisplay()` is created that simply returns success without actually moving an incorrectly positioned display to a valid location. Figure B.3 shows this function written in C, and the corresponding PowerPC and x86 assembly. In this figure, the otool output has been augmented with the hex values that represent the opcodes on disk or in memory, these are shown in the third column.

```
int
fakeCGXLFindBestPositionForDisplay() {
    return 1;
}

_fakeCGXLFindBestPositionForDisplay_ppc:
    +0  00002c24  38600001  li      r3,0x1
    +4  00002c28  4e800020  blr

_fakeCGXLFindBestPositionForDisplay_x86:
    +0  00002e23  55        pushl   %ebp
    +1  00002e24  89e5      movl    %esp,%ebp
    +3  00002e26  b801000000 movl   $0x00000001,%eax
    +8  00002e2b  5d        popl    %ebp
    +9  00002e2c  c3        ret
```

Figure B.3: Assembly dump of fakeCGXLFindBestPositionForDisplay(). (a) C code, (b) corresponding PowerPC assembly, (c) corresponding x86 assembly.

Both functions simply load a value of 1 into a register or stack location depending on the calling convention of the architecture, and then return.

To patch `CGXLFindBestPositionForDisplay()`, the existing binary code in the operating system is replaced by the hex values listed in Figure B.3.

B.2.2 vmutils

Mac OS X includes vmutils, a private, undocumented framework that allows Apple internal developers to manipulate executable binaries on disk and in memory. By using `class-dump`[1], an open-source wrapper around `otool`, header files may be generated to allow vmutils to be used as a regular public framework by 3rd party developers.

[1] `http://www.codethecode.com/Projects/class-dump/`

Once vmutils is working, the executable binary is loaded from disk or memory into an **image** object. At this point, finding the memory address of a symbol is a one-line method call in vmutils:

```
[image addressOfSymbolWithCName:
    ''CGXLFindBestPositionForDisplay''];
```

Armed with the address of the original function, the binary code at that location can now be replaced by the modified implementation developed in Section B.2.1.

B.2.3 In memory

The initial approach to patching `CGXLFindBestPositionForDisplay()` was via an in-memory patch, applying it just before calling `CGConfigureDisplayOrigin()`, and reverting to the original binary code after the call had completed. This just-in-time approach would offer the most flexibility in the face of system upgrades, and would ensure that the system remains in an untouched state for other clients of the CoreGraphics framework.

Because CoreGraphics is used by every GUI-based process, it is already mapped into the address space of the that process. An initial attempt was made to patch in this space, which failed. This was due to the `CGXL*` calls running in the WindowServer process, rather than in the patching process.

Inspecting CoreGraphics with `nm` shows that `CGXLFindBestPositionForDisplay` is a local (i.e. non-external) symbol. This means that the symbol is not available to outside callers, and makes it more difficult to determine the memory address from a client application to patch the running memory image.

Further, the virtual memory page containing `CGXLFindBestPositionForDisplay()` is marked as read-only. Initially it appeared as though it may be unlocked and patched via some virtual memory manipulation calls, but the memory page is marked as **copy on write**[5], and the page is copied to the address space of the process doing the manipulation before the changes are applied. Calling `CGConfigureDisplayOrigin()` generates an event in the WindowServer, which then is responsible for calling `CGXLFindBestPositionForDisplay()`. Since the modified virtual memory page is now a copy that exists in the patching application's memory space, the WindowServer runs the unmodified original and the patch is not utilized. This is illustrated in Figures B.4 and B.5.

Attempting to patch within the WindowServer's address space results in the same problem: unless the WindowServer itself executes the patch, the changes are not made in its process space.

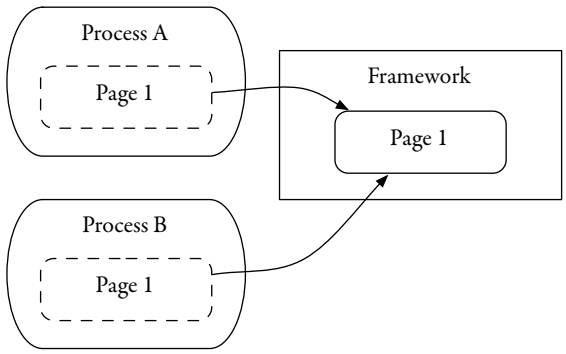

Figure B.4: Before modifying a read-only virtual memory page.

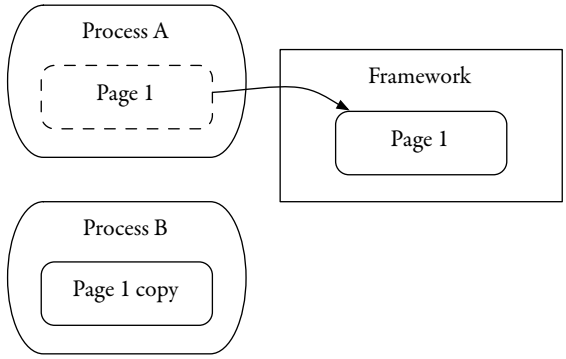

Figure B.5: After modifying a read-only virtual memory page. The page is copied to the the address space of the patching process.

Injecting a patch into the WindowServer was deemed too complicated for the purposes of this research.

B.2.4 On disk

To work around the difficulties encountered with patching CoreGraphics in memory, a file-based patch was developed. Editing the file on disk has the advantage that, after rebooting, the read-only memory image has the patch applied and available in all process spaces—no further runtime adjustments are necessary. In other situations, having the patch applied for all processes may adversely widen the scope of its impact beyond the one process that it is intended for.

There are disadvantages to this method, however. Firstly, it requires editing a core system file and rebooting, which users may be hesitant to do. Secondly, the patch must be applied whenever the file is legitimately changed by system upgrades—it's yet another thing to remember to fix after

applying an upgrade. This process may be streamlined by incorporating the patch utility into the client application to ensure that the patch is applied before the application can be used.

B.2.5 Implementing the patch

To implement the patch on disk, vmutils is used to load the file and determine the address of the `CGXLFindBestPositionForDisplay()` symbol in the PowerPC executable image.

To finish the patch, the 8 bytes of PowerPC code developed in Section B.2.1 are copied into the file at the address supplied by vmutils. This procedure is illustrated in the thesis source code, see Appendix D for details.

Note that current versions (10.4) of Mac OS X for x86, and future versions (10.5) for both PowerPC and x86, contain signed operating system binaries to prevent tampering. This will have to be circumvented for this patching method to continue working.

B.3 Conclusion

This Appendix has provided an illustrative example, using arbitrary display placement, of the types of code spelunking that were necessary to carry out this research project. In a larger context, it documents tools and techniques that are available for implementing novel functionality within the confines of a predominantly closed-source operating system.

Appendix C

Graphic Interface Specifications

Current computer displays are raster devices—each pixel on screen is addressable, and is represented in the computer by a framebuffer. Because it is impractical to provide a data line from a computer to a display for each pixel, the parallel data of a framebuffer must be serialized before it is transmitted to the display. In the case of a CRT, the data is re-converted back to its initial parallel form by the mechanics inherent to the technology. Flat panel displays digitally unserialize the data to some degree prior to final display by the low level panel driver chips.

C.1 Extended Display Identification Data

The extended display identification data (EDID) is a specification designed to provide configuration information of a sink device (typically a display) to a source device [96]. This data includes the operating parameters of the device, for example, device name, resolution, and colour capabilities.

The current version of the EDID format is enhanced-EDID (E-EDID) 2.0, the specifications of which are not freely available. After several revisions of a struct-based EDID, the E-EDID format now allows for chunk-based extensions to the base format to future-proof against new display capabilities.

Based on information in the OpenLDI specification, E-EDID 1.2 and later contain a Preferred Timing Mode bit that may be used to indicate the native or best resolution of a display device [57]. The UDI specification requires UDI sinks to advertise their native resolutions in the first display timing descriptor of the E-EDID [93]. These conflicting requirements, along with anecdotal evidence, indicate that this aspect of the EDID specification is not implemented effectively enough to achieve the desired result. Note that it is incorrect to assume that the highest advertised resolution is the native resolution of the device, as many sinks contain scaling hardware capable of downsampling as well as upsampling.

C.2 Analogue Video Data

Analogue video data for computer display use is transmitted on five channels: red, green, blue, horizontal sync, and vertical sync (RGBHV). This is commonly and incorrectly known in consumer terms as VGA, due to the graphics hardware that first popularized its use. Similarly, the connector used is properly known as HD-15, however it is widely referred to as a VGA *connector*. Of the 15 total pins in the connector, 10 are used for RGBHV data, and three are used for EDID.

C.2.1 Bandwidth

Current consumer analogue displays typically support a maximum bandwidth of 350 MHz, spread across all five channels. Each full-colour pixel requires a discrete value for the red, green, and blue pixel elements (pels), and each pel takes one clock cycle to transmit. Equation C.1 estimates the maximum number of pixels a given device may support based on the total link bandwidth. Because the horizontal and vertical sync channels don't require as much bandwidth as the RGB data, they are ignored for the purposes of this calculation. However, it should be noted that between each frame is a blanking interval, which was originally designed to give the electron beam in a CRT time to return to the origin. In the RGBHV format, this interval is a fixed duration, so it consumes a greater percentage of the total available bandwidth as the pixel clock rate increases [22].

$$350\ \text{MHz} \times \frac{1\ \text{s}}{60\ \text{frames}} \times \frac{1\ \text{pixel}}{3\ \text{channels (RGB)}} = \frac{1.94\ \text{Mpixels}}{\text{frame}} \tag{C.1}$$

The common 4:3 aspect ratio resolution of 1600 × 1200 uses 1.92 Mpixels per frame, which utilizes 345 MHz of bandwidth at 60Hz. Professional equipment can support resolutions up to 2048 × 1536 with approximately 500 MHz of bandwidth. The Matrox TripleHead2Go has an input resolution of 3840 × 1024, using bandwidth in excess of 700 MHz [47]. Because RGBHV is an analogue format, the upper bandwidth limit is not explicitly fixed by the format specifications, but is implicitly determined by cable and component quality.

C.3 Digital Visual Interface (DVI)

DVI is currently the most popular digital graphics interface for computer display use, and is the basis for the video portion of the HDMI and UDI formats.

C.3.1 Bandwidth

The digital portion of DVI supports one or two links, and each link is comprised of three channels, one each for red, green, and blue. Because the link sends the RGB triples simultaneously, much lower signalling speed is necessary to transmit the same amount of data as an equivalent RGBHV connection. Resolutions up to and including 1920 × 1200 are supported on a single-link capped to 165 MHz. Dual-link DVI, which transmits two pixels per clock, is provided for resolutions and refresh rates requiring more bandwidth than the single-link can provide. Dual link bandwidth is not capped by the specification, and is only restricted by cable and component quality. Consumer displays with resolutions of 2560 × 1600 are currently being driven with a DVI-DL interface.

When driving a digital flat panel display or other technology that does not require CRT beam synchronization between frames, DVI may use a reduced LCD blanking, which requires only 5 per cent of the total clock, rather than a fixed duration [22]. This leaves much more bandwidth for image data at high pixel rates.

C.3.2 Analogue

DVI is unique among digital graphics interfaces, because it may also include an analogue RGBHV link. DVI-A provides only the analogue link, DVI-D is digital only, and DVI-I integrates both an analogue and a digital link. Example DVI connectors are shown in Figure C.1. This allows for the creation of a *gender bender* adaptor, which features a DVI-I connector on one end, and an HD-15 connector on the other. This is a popular way to allow current source devices with DVI-I outputs to utilize RGBHV sinks. The analogue component of DVI is specified to support a bandwidth of 400 MHz.

C.3.3 Other features

As currently implemented, DVI transmits the entire image each time the screen is refreshed. However, displays are growing in logical size faster than copper interconnect speeds are increasing, and this model will eventually lead to starvation. To counter this, the DVI protocol designers have made provisions for selective refresh, in which only frames that have changed are sent [22].

When used in dual-link mode, two pixels are sent per clock. Link 1 transmits odd pixels, and link 2 transmits even pixels. Scan lines start on pixel one.

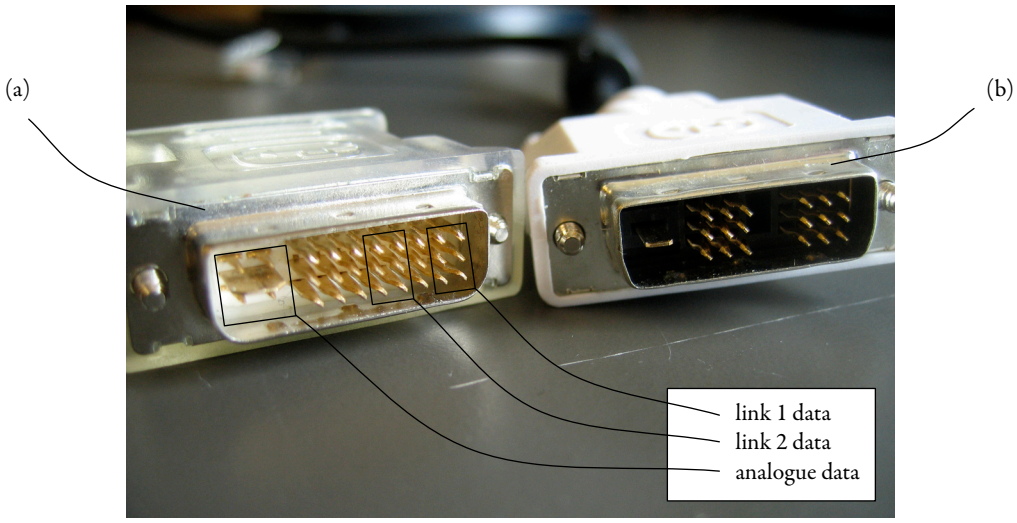

Figure C.1: Different DVI connectors. In a DVI-I (dual-link) connector (a), both analogue and digital signals are supported. In a DVI-D connector (b), the connector is missing the four analogue pins surrounding the large flat pin, and the central six pins for the second digital link.

C.4 High-Definition Multimedia Interface (HDMI)

HDMI is an integrated digital audio and video interface, designed for home theatre use. HDMI used DVI as the basis for the video portion of the link; however, in recent revisions it has diverged from the DVI specification somewhat. Notably, HDMI supports different colour formats (RGB 4:4:4, YCbCr 4:4:4, and YCbCr 4:2:2), increased bit depth (24-48 bits per pixel), and increased signalling frequencies on both single (340 MHz) and dual (680 MHz) link versions [33].

While revision 1.3a of the HDMI makes provisions for substantially increased data rates over those of DVI, it should be noted that this bandwidth must be shared with the audio data. In addition, the HDMI specification has not yet defined any resolutions with a total number of pixels greater than 1920×1080 (1080p).

C.5 Unified Display Interface (UDI)

Like HDMI, UDI is an extension to DVI. Rather than adding audio capabilities, UDI augments DVI with further high resolution and high depth video capabilities [93]. UDI is offered in both an embedded (for laptops or other devices with built-in displays) and external version.

In the external profile, UDI utilizes three lanes to transmit image data, one each for colour in

an RGB pixel. One byte or *symbol* is transmitted per lane per clock, thus one pixel is transmitted per clock in in the 24 bits per pixel format. Higher and lower bit depths are supported by packing pixels and running the symbol rate at a direct ratio of the pixel clock.

C.6 Low Voltage Differential Signalling (LVDS)

LVDS is often incorrectly used to refer to the display interface in a laptop or other computer with a fixed flat panel. LVDS is analogous to the transition minimized differential signalling (TMDS) digital data transport technology employed by DVI and its derivatives. Two higher level protocols that use LVDS to interface flat panel displays are OpenLDI and FPD-Link [57, 56]. As these links are quite specialized and used to drive displays that are physically restricted in size, their resolution specifications will not be discussed here.

Unlike DVI, dual link OpenLDI divides the screen vertically in two, and sends the top and bottom halves simultaneously.

C.7 DisplayPort

The Video Electronics Standards Association (VESA) have introduced DisplayPort, a competing standard to DVI and its derivatives [95]. Because DisplayPort shares no common ancestry with the DVI derived formats, it is not capable of interoperating with those standards. In addition, version 1.0 of the specification used DisplayPort content protection (DPCP) rather than the more prevalent high-bandwidth digital copy protection (HDCP) used by DVI and its derivatives. DisplayPort version 1.1 adds support for HDCP.

Unlike the other formats, DisplayPort acts more like a generic digital bus than a specialized graphics transport: the link rate is decoupled from the pixel rate, and pixels are timestamped, padded, and packetized before being placed on the bus. The Main Link, responsible for carrying video data, is comprised of 1-4 *lanes* which transmit pixels in an interleaved fashion, similar to the approach used by DVI-DL. The Main Link may operate at two speeds and use 1, 2 or all 4 lanes to achieve the necessary bandwidth to carry the requested pixel format and image size.

In an effort to support multiple displays driven from a single graphics card, VESA have sized the DisplayPort connector so that 4 will fit within standard ATX/BTX bracket opening used by AGP, PCI, and PCI-E add-in cards.

C.8 Summary

Table C.1 summarizes the similarities and differences between the video interface standards outlined above. Recall that for an analogue interface (VGA, DVI-A), one pel is transmitted per clock, requiring 3 clocks for an RGB pixel. At 24 bits per pixel, the digital formats transmit at least one whole pixel per clock. At higher bit depths, they may utilize a faster symbol clock or more links to transfer a pixel.

The stream-based analogue RGBHV interface heavily influences the data transfer paradigm of DVI and its derivatives—pixels are transmitted one after the other in linear scan lines. DisplayPort is the first graphics interface to break from this mould, and while the current implementation is also stream-based, the architecture supports a more general data transport mechanism that may one day see an entire frame compressed and packetized at the frame level, rather than at a per-pixel granularity.

Table C.1: Comparison of video standards

	Max. Res.*	Max. Bwidth	Conf. Protocol	DRM
VGA (RGBHV)	3.9	350+ MHz	E-EDID 2.0	none
DVI-A	3.1	400 MHz	E-EDID 2.0	none
DVI 1.0	2.6	165 MHz	E-EDID 2.0	HDCP
DVI-DL 1.0	5.8	350+ MHz	E-EDID 2.0	HDCP
HDMI-A 1.3	2.1	340 MHz	E-EDID 2.0	HDCP
HDMI-B 1.3	2.1	680 MHz	E-EDID 2.0	HDCP
LVDS (OpenLDI)	3.1	160 MHz	E-EDID 2.0	none
DisplayPort 1.1	6.9	10 Gbps	E-EDID 2.0	DPCP & HDCP
UDI	11.1**	16 Gbps**	E-EDID 1.3	HDCP

* Maximum resolution in megapixels at 60 Hz refresh, 24 bits per pixel.
* Estimated.

Appendix D

Project Source Code

The source code developed for this thesis may be found online.

- `http://code.google.com/p/reflect`
- `http://code.google.com/p/satellitedisplays`
- `http://code.google.com/p/ioproxyvideofamily`

Appendix E

Current Developments

At the time of this writing, Microsoft's new operating system, Windows Vista, has just been released. Every effort has been made to follow the development of this system and keep abreast of the technological advancements over its predecessor, Windows XP, and the impact they may have on the work presented here. It should be noted that the specifications for Windows Vista were quite volatile during its development, so technical information available prior to release was often informal, incomplete, and subject to change.