Graz University of Technology

Institute for Computer Graphics and Vision

Dissertation

WIMP Interfaces for Emerging Display Environments

Manuela Waldner

Graz, Austria, June 2011

Thesis supervisor

Prof. Dr. Dieter Schmalstieg

Referee

Prof. Dr. Andreas Butz

To Martin

Abstract

With the availability of affordable large-scale monitors and powerful projector hardware, an increasing variety of display configurations can be found in our everyday environments, such as office spaces and meeting rooms. These emerging display environments combine conventional monitors and projected displays of different size, resolution, and orientation into a common interaction space. However, the commonly used WIMP (windows, icons, menus, and pointers) user interface metaphor is still based on a single pointer operating multiple overlapping windows on a single, rectangular screen. This simple concept cannot easily capture the complexity of heterogeneous display settings. As a result, the user cannot facilitate the full potential of emerging display environments using these interfaces.

The focus of this thesis is to push the boundaries of conventional WIMP interfaces to enhance information management in emerging display environments. Existing and functional interfaces are extended to incorporate knowledge from two layers: the physical environment and the content of the individual windows. The thesis first addresses the technical infrastructure to construct spatially aware multi-display environments and irregular displays. Based on this infrastructure, novel WIMP interaction and information presentation techniques are demonstrated, exploiting the system's knowledge of the environment and the window content. These techniques cover two areas: spatially-aware cross-display navigation techniques for efficient information access on remote displays and window management techniques incorporating knowledge of display form factors and window content to support information discovery, manipulation, and sharing.

Results of user studies indicate that environment- and content-awareness of WIMP interfaces indeed improves information management in many varieties of emerging display environments. Awareness of the environment and the window content thereby can be facilitated without an obtrusive and expensive hardware and software infrastructure.

Keywords. WIMP Interfaces; Multi-Display Environments; Irregular Displays; Large-Scale Displays; Information Management; Human-Computer Interaction; Computer-Supported Cooperative Work; Window Management; Information Presentation; Cross-Display Navigation; Windowing Systems

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Acknowledgments

There is a large number of people without whom this thesis would not have been possible and many people that directly or indirectly supported my work through technical, scientific, or personal support. First of all, I want to acknowledge my supervisor Dieter Schmalstieg who gave me the opportunity to join his research group at Graz University of Technology. I particularly want to thank him for letting me pursue my own goals – despite often going far away from the actual research agenda of this institute – while still guiding and supporting my work. I also want to thank my referee Andreas Butz from Ludwig-Maximilian University in Munich for his spontaneous and uncomplicated agreement to support this thesis and our precedent discussions at conferences.

In addition, I want to acknowledge all my colleagues of the *Deskotheque* project. First of all, I want to thank Christian Pirchheim for teaching me so many things about Linux, for the countless shell scripts, for doing so much "dirty" work in our lab, and for endless discussions – not necessarily limited to the project. I also want to point out the short but fruitful collaboration with Werner Puff who made valuable contributions to the visual links infrastructure and was also a great help for porting the Deskotheque framework. Finally, Markus Steinberger was more than just a "coding monkey" for this project. Still, I first want to point out his stunning ability to implement high-performance solutions to complex optimization problems in minimum time. I also want to thank him for his general support in the last year of the project.

Apart from the project members, there are many other colleagues at ICG that deserve being mentioned at this point, for discussions and support: Alexander Lex and Marc Streit, Albert Walzer, Raphael Grasset and Ernst Kruijff, Mark Dokter, Markus Murschitz, Werner Trobin, Matthias Rüther, and Christopher Zach, Erick Mendez, Markus Tatzgern, Andreas Wurm, and many more. A special thank, of course, goes to all the participants of our user studies, their patience and all their valuable feedback! This is probably also the right spot to also acknowledge all the Linux open source developers – especially the Compiz and multi-pointer X developers.

Finally, I want to thank my family who made this work possible by believing in me all these years. I want to gratefully mention my partner Martin for always being there for me, for all his support and endless patience. The least I can do is dedicate this thesis to you.

Contents

Part I

Overview

Chapter 1

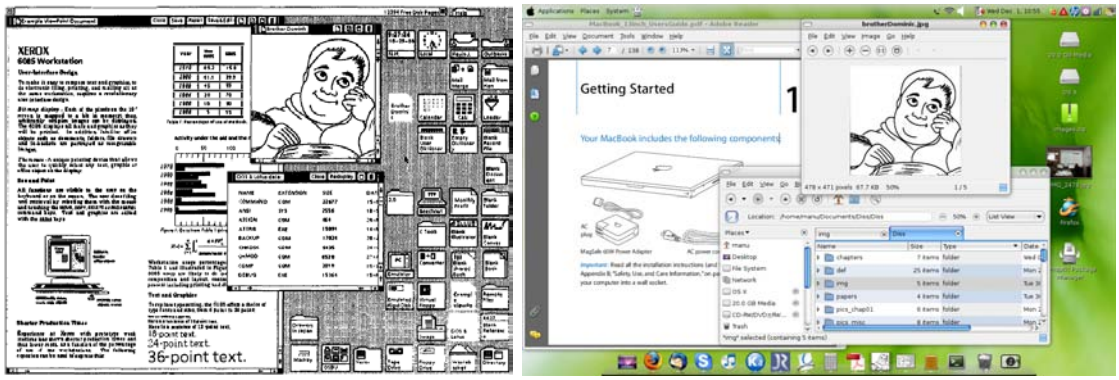
Introduction

In recent years, display hardware has become more affordable and more powerful, in terms of resolution, size, and brightness. As a result, we can see a variety of display configurations in our everyday working environments, living rooms, meeting rooms, control rooms, and public spaces. Examples range from already quite common dual monitor configurations connected to a work PC to complex control room setups combining multiple personal and shared display spaces into a common interaction space. Of particular interest are office environments leveraging projection technologies to augment multiple isolated single-user workspaces to a shared interaction space (Figure 1.1).

Figure 1.1: Multi-display environments combine projected displays and monitors of different size, resolution, and orientation into a common interaction space.

Yet, the WIMP (windows, icons, menus, and pointers) metaphor has remained largely unchanged since its introduction more than 30 years ago. The Smalltalk system (1979) [89] and the Xerox Star (1981) [132] were the first systems allowing users to show multiple pieces of information (*views* [89]) simultaneously on the screen (Figure 1.2(a)). Today, WIMP interfaces mainly rely on the same fundamental concepts: they provide multi-

ple overlapping, rectangular *windows* to physically separate different applications on the screen (Figure 1.2(b)) and a single *pointer* to access and manipulate their content. The internal screen representation – describing either a single or multiple output devices – of WIMP interfaces has hardly been adapted to the emerging display factors and is still described as simple rectangle defined by the overall number of pixels.



(a) (b)

Figure 1.2: WIMP interfaces 1981 and now: (a) the desktop metaphor of the Xerox Star [132] and (b) of a modern Linux (Ubuntu) operating system.

To access and manipulate the window content, as well as to lay out the individual windows on the screen, indirect pointing devices, like mouse or trackball, are still the most common input devices. Although a variety of input technology is now available for emerging displays, like touch-sensitive surfaces, digital pens, or some sort of embodied interaction, the mouse is still a valuable universal controller for heterogeneous displays. However, as displays are getting larger and more discontinuous (like in Figure 1.1), mapping the incoming 2D motion events to a specific location in the environment is becoming more challenging. In today's systems, the screen space – consisting of a single or multiple displays – is mapped onto a simple rectangle, which defines the boundaries of the navigation space. In an environment like in Figure 1.1, a simple rectangle cannot capture the complexity of the display setup and may lead to an unintuitive navigation space.

As a result, new resources and possibilities emerging from new display hardware remain largely unused in conventional working environments, as a significant cognitive effort is required to use these mature, but maybe no longer appropriate, WIMP interfaces in such an environment. The focus of this thesis therefore is to enhance information management with WIMP interfaces in emerging display environments – in particular in office spaces – by addressing three major areas:

1. Designing an infrastructure for interactive workspaces, extending multiple isolated single-user workspaces to a shared interaction space,

2. exploring multi-display navigation techniques for the conventional computer mouse

for effective information access in discontinuous display environments, and

3. extending window management techniques to improve information discovery, manipulation, and sharing in such an interactive workspace with conventional application windows.

1.1 Emerging Display Factors

With the introduction of low-cost high-resolution monitors, self-contained devices, and projectors, new display form factors and arrangements have evolved:

(a) (b) (c)

Figure 1.3: Emerging display factors: (a) large, ubiquitous displays, (b) non-planar, non-rectangular displays, and (c) horizontal displays.

- Monitors have a higher number of pixels than a few decades ago. In addition, multiple monitors or projectors can now be combined to a large display space, resulting in a display surface of several megapixels, or even gigapixels. When using projectors, the boundaries of these large displays may be visually dissolved, leading to ubiquitous information spaces (Figure 1.3(a)).
- Projectors may be mounted to cast images onto an irregular, non-planar surface. Oblique projection angles or combinations of multiple projectors additionally cause non-rectangular outlines (Figure 1.3(b)).
- While the number of pixels generally increases, the resolution of displays may strongly vary. Projected displays usually have a much lower resolution than monitors and are located at a greater distance from the user. Therefore, projectors are rarely employed for private work but are rather suitable for public information spaces, observed and operated by multiple users.
- Projectors easily enable the creation of non-vertical displays, such as tabletop displays or tilted surfaces (Figure 1.3(c)).
- Finally, all of these display factors may be arranged in a common environment, resulting in a heterogeneous interaction and viewing space (Figure 1.1).

This list is not exhaustive but aims to capture the most important properties with respect to office environments. Particular properties of very small, portable devices or devices with direct input capabilities will not be addressed in this thesis.

Definitions

In the context of display environments, the following terms will be used in this thesis:

An *output device* refers to a single monitor or projector, *i.e.*, a single physical display entity connected to a single PC. In this thesis, a *display* is defined as a continuous output, such as a single monitor or projected imagery – either produced by a single projector or by multiple projectors. A display composed of multiple, seamlessly aligned projected images will be referred to as *tiled display* or *multi-projector display*. The resulting display from a projector mounted towards a non-planar surface will be called *irregular display*. A horizontal display will be called *tabletop display*.

A *multi-display environment* (MDE) combines multiple discontinuous, heterogeneous displays into a common virtual space. In this thesis, MDEs will be discriminated from *multi-monitor environments*, which consist only of monitors, which are often heterogeneous. The reason for this separation stems from the fact that multi-monitor environments are already becoming ubiquitous and the motivating scenario for this thesis evolves mainly around heterogeneous environments, combining multiple monitors with projected displays (irregular and / or tiled). MDEs can be driven from a single or from multiple machines (*distributed MDE*). Each machine in this distributed environment has one *screen*, which is the entity of virtual pixels assigned to one or multiple connected output devices. The case of multiple virtual screens, by using multiple *virtual desktops* [70], will be disregarded in this thesis.

One in general ambiguously used term is *resolution*. In this thesis, we will refer to resolution as the pixel density, *i.e.*, the resulting pixels per inch on the display. Resolution will not be used to describe the number of pixels of a screen or output device.

1.2 WIMP Interfaces

Windows are *views* of some particular data or application in the computer [51, 89]. They visualize data, such as text, graphics, or videos [162], usually within a rectangular boundary. A *window manager* is responsible to manage multiple of such windows on the screen [162] and to forward input from a specific device to a particular window. Window managers are special management entities of *windowing systems* (like the *X Window System* [211]), which provide the windows themselves, as well as low-level rendering capabilities and input handling [163]. On top of the windowing system, (GUI-)*toolkits* provide common user interface widgets, like menus or buttons [163].

The term *WIMP (windows, icons, menus, pointers) interfaces* is commonly used to describe the currently most wide-spread type of "point-and-click" graphical user inter-

faces [248]: multiple windows embedded in a *desktop environment* with icons or panels, operated by an indirect pointing device like the mouse. The focus of this thesis lies on window management and (mouse) pointer interaction. Menus and icons are not explicitly addressed.

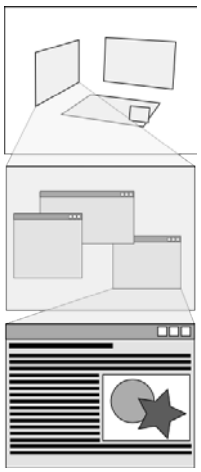


Figure 1.4: WIMP interface layers.

With respect to WIMP interfaces, we will discriminate the following three interface layers in this thesis (Figure 1.4):

- The top **environment** layer contains a single or multiple displays of various form factors, which are somehow spatially arranged towards each other, as well as one or multiple users.
- The **screen** layer is the virtual representation of one of these displays. Each screen contains multiple windows, which are spatially arranged on the available screen space, together with other desktop elements, such as icons and menus.
- The bottom **window** layer deals with the content of the window. A window may contain different kind of visual information, like text, images, charts, or videos.

Classic WIMP interfaces manage the second layer and also derive their knowledge only from the screen layer. Window managers deal with the initial window placement when a window is mapped and provide interaction techniques for manual window repositioning and resizing with pointing devices, like the mouse. In addition, they provide slightly more sophisticated features like *maximize*, which resizes the window to fit the margins of the display – often even aware of multi-monitor seams. However, classic WIMP interfaces are unaware of the windows' content and the surrounding context by the physical environment, such as the spatial arrangement of multiple displays and the users in the environment.

1.3 Information Management

Modern information workers need to explore large information spaces to reach crucial decisions. With modern processing powers and the availability of a massive amount of data in the world-wide-web, information workers need to conduct a number of steps to extract knowledge from the available information sources. According to the sensemaking model (*e.g.*, [53, 102, 206]), information workers first need to forage for data (*i.e.*, gaining an overview, browsing, searching), then create a schema (*i.e.*, operating on the data and synthesizing results), to finally solve a problem, make a decision, and act. Pirolli and Card [182] separate the overall sensemaking process into two major loops (Figure 1.5): The *foraging loop* includes processes of information search, filtering, reading, and searching for relations. In the subsequent *sensemaking loop*, the user schematizes the collected information, generates hypotheses, and communicates the findings. From the final step,

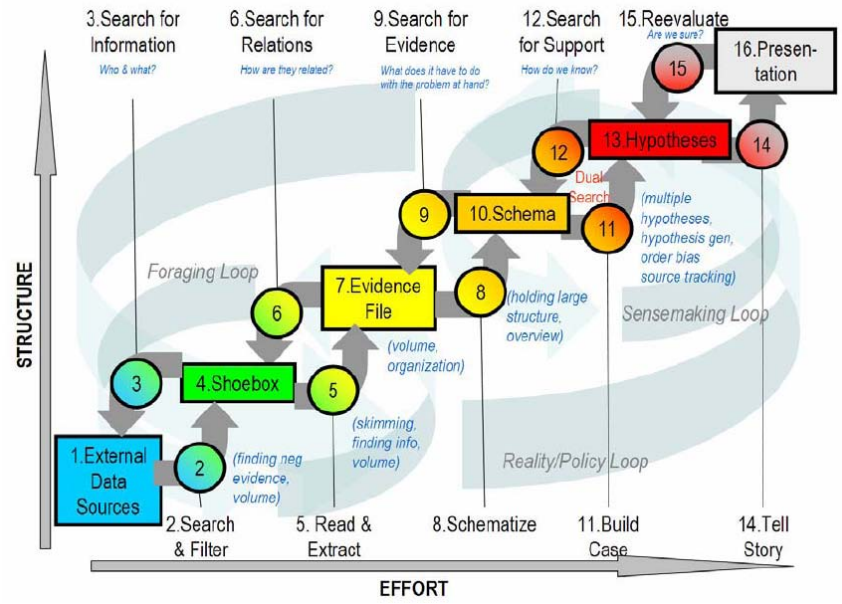


Figure 1.5: The sensemaking loop by Pirolli and Card (from [182]).

the user may then re-evaluate the findings and search for evidence, which leads back into the foraging loop.

There is evidence that emerging display form factors support users in their sensemaking process. Large displays may act as *external memory*, helping users to memorize and organize information items [4]. Physical discontinuities caused by monitor bezels can help users to partition their information [91]. In addition, emerging display form factors, such as tabletop displays or large wall displays, may enhance co-located collaboration [74, 215], which is known to improve quality and performance of information extraction and decision making tasks [104].

Even though the display form factors potentially support users in their sensemaking process, WIMP interfaces, which are not adapted to these display form factors, can hamper the user in her sensemaking process. The following examples will illustrate these deficiencies with respect to specific activities: information discovery, access, manipulation, and sharing.

1.3.1 Information Discovery

With a wealth of information sources available, the user has to search for particular information items contained in multiple sources. She might look for a specific object or she may conduct an exploratory, open-ended search concerning a particular topic. The information of interest may be scattered in different application windows in the environment, together with information, which is not relevant for the current task.

With increasing display space, more information can be visualized concurrently. How-

ever, relevant information on the screen can also be invisible for the user, even if there is sufficient display space to arrange multiple non-overlapping windows. With increasing display space, users simply have more application windows open [112]. Bezerianos *et al.* [31] list a number of possibilities why information may be hidden, for instance due to an occluder window, physical occlusions, or because the information is outside the user's visual field of view. Inspired by this list, we categorized these possibilities into the three layers of WIMP interfaces (*cf.*, Table 1.1).

	Window	**Screen**	**Environment**
Occluded	tabs	overlapping windows	physical occlusion
Cropped	window boundary	display boundary	field of view
Diminished	zoomed out	minimized, iconified	too distant

Table 1.1: Invisible information on window, display, and environment level: information can be occluded, cropped, or too small or abstracted to be perceived (diminished).

Simple occlusion management techniques [73] are available in nowadays' window managers – for instance as *Exposé* or Alt+tab menu (Figure 1.6). However, these techniques operate on a screen level only and neither take the windows' content, nor the display factors into account. For instance, the Alt+tab menu of Figure 1.6 hardly helps users to discover a specific window on a very large display, if windows are arranged side-by-side, and does not reveal information outside a window's visible boundary. The simple occlusion management techniques of current window managers also do not help users to filter a large amount of visible information.

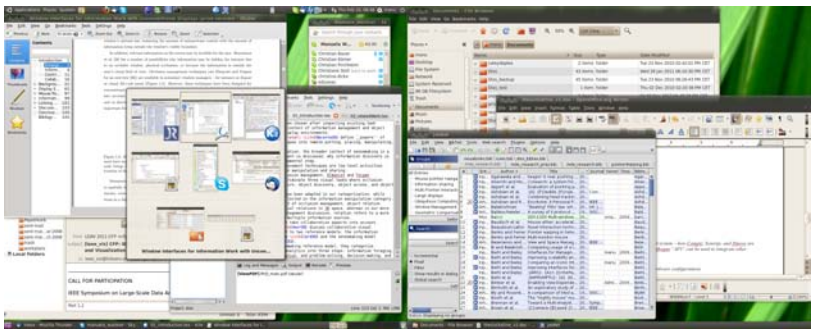


Figure 1.6: Alt+tab on a conventional dual-monitor screen: Switching through the application thumbnails brings the individual windows to the front, but there is no visual cue about the location of the currently selected window.

With classic WIMP interfaces, the user therefore has to undertake a series of actions to discover information: She has to spatially lay out multiple application windows, restack windows to reveal occluded elements, resize windows to discover cropped elements, and physically navigate to locate items in a distance. Future window management techniques should provide effective occlusion management techniques and attention guiding mechanisms to help the user discover invisible information on all levels and to filter a large amount of visible information.

1.3.2 Information Access

Once a desired piece of information is discovered, the user may want to access or select it for further manipulation or transfer to a remote location. Users need to resolve potential occlusions and maybe relocate the item of interest to a convenient location. In the simple example of the Alt+tab menu in Figure 1.6, the user linearly steps through the menu until reaching the anticipated target window. Releasing the keyboard shortcut brings the window to the front, *i.e.*, it resolves the occlusion on the screen level. To access the window content with the mouse, she first needs to move the cursor on top of the window. On a small-scale monitor this often does not require any additional activity.

However, in a large display environment, accessing information often leads to either extensive virtual or physical navigation. Users may either move their cursors to a remote display space or physically walk or reach to a distant location. In heterogeneous, discontinuous display environments, accessing an application window's content with the cursor leads to considerable user interface challenges: The user may have to overcome extensive distances between the current focus object and the anticipated target location – either by crossing a large number of pixels or by overcoming display-less space between displays. In addition, combining displays of diverging form factors leads to a heterogeneous navigation space with respect to resolution, orientation, size, and distance to the user. Visibility issues, such as readability problems of distant information or distortions caused by oblique viewing angles, may necessitate the user to change physical location, to transfer the item closer, or to adjust the window content's zoom level.

WIMP interfaces for emerging display environments therefore do not only need to optimize navigation in the environment in terms of travel distances and mental effort, but may also consider unifying window manager functions and navigation tasks for more efficient information access.

1.3.3 Information Manipulation

Once an item is selected, the user can apply more fine-grained operations, such as extracting detailed information, operating on the data, or comparing items across multiple objects. On the screen level, a large screen space allows the user to freely arrange and organize separate pieces of information, represented by individual application windows, resulting in a large visual search space. She can place pieces of information next to each other for comparison, maximize windows to focus on a single item, deposit items for later investigation, or spatially organize items for a better overview. However, window managers usually leave the spatial window arrangement and window resizing to the user. This results in a considerable management overhead for the user when dealing with multiple information sources. To user has to frequently move windows and to carefully select the window's optimal size, balancing the amount of unimportant content with the amount of information lying outside the window's visible boundary. Early experiences with window managers [50] have shown that having multiple windows helps users to finish a task more

accurately, but managing the windows introduces so much effort that users are actually faster with non-windowed interfaces. More recently, it has been shown that users have more open application windows if the available display space increases [112]. This leads to an increased amount of time spent for window management on large-scale displays, compared to conventional monitors [32].

Future window management techniques may support users in creating an optimal spatial window layout by automatically increasing information visibility or by transferring items to suitable locations, thereby exploiting inherent knowledge of the spatial environment configuration. Also, information management across multiple windows can be facilitated by providing special side-by-side comparison arrangements or by attaching related windows to each other, without introducing additional occlusions.

1.3.4 Information Sharing

Solving a complex task often requires the expertise of multiple users with different knowledge backgrounds. In large display environments, sufficient screen space, ample space in front of the display, and physical partitions help users to arrange their territories and to lay out information sources in different, user-specific visual encodings.

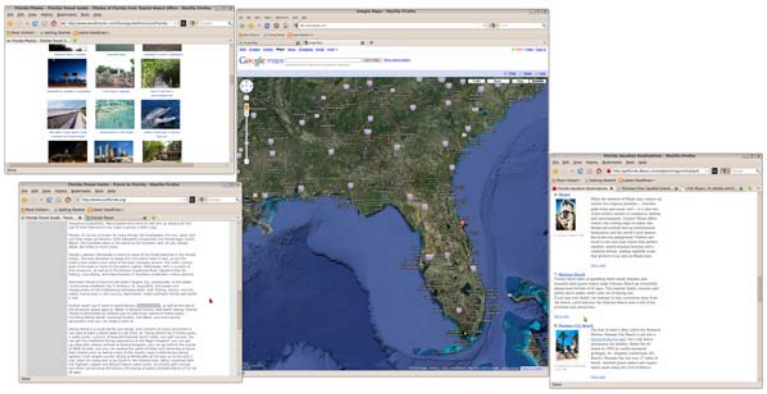


Figure 1.7: Two users interacting on a shared display: although the pointers are color-coded, they are hard to spot and to discriminate (located in the two lateral browser windows on the lower display half). On a distant display, this makes deictic referencing challenging.

Current WIMP interfaces do not support such a co-located collaborative information work in a sufficient manner. Crucial underlying requirements, such as multi-pointer support and access management, have hardly found their way from research prototypes to fully functional window management features. This lack of support impedes parallel work in shared display environments, but is also hampering communication and an equitable workload, as usually one user is taking control.

Apart from these fundamental requirements, window managers do not provide the necessary coordination techniques to support successful collaboration (*cf.*, the *mechanics of collaboration* [95, 179]). Coordination techniques may provide subtle awareness of

other users' activities, support for obtaining, reserving, and handing over resources. In particular, window managers for large display environments may visually highlight items which are currently under other users' investigations, thereby providing basic workspace awareness and supporting deictic referencing during discussions. Consider, as an example, Figure 1.7: even with sufficient multi-pointer support, synchronous interaction is challenging, as there is no visual support for multi-user activities across multiple application windows.

In addition, interaction techniques are required to take over items from a collaborator without disturbing the others' workflow and to easily protect workspace items from other users' activities. When sharing information with a collaborator, objects are often moved to a display location where all involved team members can easily view and, if necessary, operate on the data. As the users return to individual investigations again, the content is transferred back to private display spaces, where uninterrupted work should be possible. Future window managers should support users in simple window transfer by exploiting their knowledge of the environment's configuration.

The four above listed activities of information management in WIMP interfaces (discovery, access, manipulation, and sharing) were chosen after careful inspection of related work in the field, such as the sensemaking model [53, 102, 182, 206]. In more specific contexts, other categorizations are used. In the context of occlusion management interfaces, Elmqvist and Tsigas [73] discriminate three visual tasks where occlusion management usually occurs: object discovery, object access, and object relation. The first two tasks have been adapted in our categorization, while object relation is reflected in the information manipulation category.

In an even more specific taxonomy, Nacenta *et al.* [166] define "powers" of object movement techniques in MDEs into remote putting, placing, manipulating, and getting. These powers of object movement techniques are low-level activities embedded in information access, manipulation, and sharing in our activities.

For window management, Kandogan and Shneiderman [135] discriminate three tasks: environment setup, task environment switching, and task execution. Windows are assumed to be uniquely associated with a single task, while the user has to frequently switch between tasks in the course of her work. Task execution is furthermore split into sequential scanning of multiple windows for a certain information item, comparing multiple sources based on the identified items, determining the context by filtering the items and proceeding to scan, and recalling previous items. While task management and switching is a very important topic in window management and computer-supported information management in general [15, 52, 67], it is not reflected in our activities. We rather concentrated on single task execution, such as scanning (information discovery), comparing and filtering (manipulation).

1.4 Research Hypothesis

These previous examples demonstrate that current WIMP interfaces do not adequately support users in information management in emerging display environments. Therefore, such environments are usually equipped with radically new user interfaces explicitly tailored towards the emerging display factors (for an overview, refer to Section 2.2). In contrast, this thesis aims towards a different goal: pushing the boundaries of existing WIMP interfaces to break out of the single-user, single-display box.

The research question addressed in this thesis evolves around information management in office environments. Single-user workspaces are extended to collaborative interactive spaces using off-the-shelf projection technology. WIMP interfaces and interaction techniques are facilitated and extended for improved information management.

The basic hypothesis is: *Incorporating knowledge of the physical environment and window content in WIMP interfaces will support users in discovering, accessing, manipulating, and sharing information.*

To test this hypothesis, user interface prototypes have been developed on top of a spatially aware multi-display framework facilitating compositing window managers and recently established multi-pointer technologies. These interfaces have been evaluated in laboratory experiments – either as controlled comparative studies or as exploratory investigations.

1.5 Contribution

We addressed the goal to facilitate knowledge of the physical environment and the window content in WIMP interfaces on two levels:

1. the design and implementation of the technical foundations for awareness of the physical environment and the window content in WIMP interfaces and

2. the extension of existing WIMP interfaces to facilitate this knowledge and evaluation of their impact on information management activities in emerging display environments.

Extensions and add-ons to existing system infrastructure enabled the creation of fully functional multi-display WIMP interfaces and interaction techniques, based on a spatially-aware multi-display framework. In this way, we could properly design and evaluate our prototypes for "real" information management usage.

In the following, an overview of the individual contributions in this thesis will be presented:

- Creation of a multi-display framework for the construction of multi-planar multi-projector displays in a low-cost fashion, which allows the employment of legacy applications in a conventional desktop environment with interactive frame-rates.

The main distinguishing aspect is an automatically created fine-grained model of the environment, which can be exploited by different components of the multi-display environment.

- Development of different cross-display mouse pointer navigation techniques for efficient information access, automatically derived from the spatial model of the display environment. An experiment comparing different spatial configurations for cross-display pointer navigation was conducted and follow-up studies were employed to clarify some of the exploratory findings in controlled experiments.
- Design and implementation of window management techniques for emerging display factors, exploiting knowledge of window content and display factors, for more efficient and satisfactory information discovery, manipulation, and sharing. The following techniques have been designed:
  - Visual links connect related information contained in multiple application windows across window and display boundaries for multiple users to improve information discovery and sharing. Exploratory experiments have been conducted to assess the efficiency of information discovery using visual links across application windows and to evaluate the effectiveness and user satisfaction of collaborative information discovery and sharing on a large, shared display.
  - Uncovering windows is a window management function that analyses the visual window content to temporarily maximize the amount of visual information on the screen for more efficient information discovery and access. An experiment was conducted to compare window uncovering with conventional window switching techniques.
  - Display-adaptive window management adapts the spatial window layout to the prevalent display form factor for easier information manipulation on irregular displays. An exploratory experiment compared selective display-adaptivity features to window management without any spatial awareness on an irregular display.
  - Polarization-based interfaces support information comparison of two spatially registered information layers using a light-weight tangible magic lens interface.

1.6 Collaboration Statement

The following publications (in chronological order) were published in the course of this thesis and serve as foundation for the thesis text:

- Manuela Waldner, Michael Kalkusch, Dieter Schmalstieg, **Optical Magic Lenses and Polarization-Based Interaction Techniques**, In Proc. Eurographics Symposium on Virtual Environments and Immersive Projection Technology Workshop

(IPT-EGVE), pages 61–68, July 2007.
Summary: This paper introduces the concept of a purely optical magic lens user interface which relies on polarization properties to separate information layers on a single shared display.
Contribution: idea and concept (together with Martin Braun); implementation of simple calibration routine; hardware setup; application scenarios (partially); content creation.
References: Section 8.4.

- Manuela Waldner, Christian Pirchheim, Marc Streit, Dieter Schmalstieg, **Multiple View Visualization On A Multi-Display Setup**, In International Workshop on Giga-Pixel Displays & Visual Analytics (GIANT), April 2008.
Summary: This poster illustrates the concept of using multi-display environments to physically separate multiple coordinated views.
C*ontribution:* main contributor for idea and concept, support for hardware setup and content creation.

- Manuela Waldner, Christian Pirchheim, Dieter Schmalstieg, **Multi projector displays using a 3D compositing window manager**, In Proc. Workshop on Immersive projection technologies / Emerging display technologiges (EDT-IPT), pages 1–4, August 2008.
Summary: This paper presents warping and blending for displays combined by multiple casually aligned projectors on multi-planar surfaces, implemented on the window manager level.
Contribution: main contributor for design and implementation of calibration routine; support for window manager integration.
References: Section 4.1 (spatial model creation), Section 7.3.2 (window manager integration).

- Christian Pirchheim, Manuela Waldner, Dieter Schmalstieg, **Deskotheque: Improved Spatial Awareness in Multi-Display Environments**, Proc. of IEEE Virtual Reality Conference (VR), pages 123–126, March 2009.
Summary: In this paper, the technical infrastructure and the creation of the spatial model of the multi-display environment "Deskotheque" is presented.
Contribution: main contributor to concept of spatially aware multi-display environment; design and implementation of seamless navigation; minor contributions to software framework and hardware setup.
References: Section 3.2 (distributed software infrastructure).

- Manuela Waldner, Alexander Lex, Marc Streit, Dieter Schmalstieg, **Design Considerations for Collaborative Information Workspaces in Multi-Display Environments**, Proc. of Workshop on Collaborative Visualization on Interactive Surfaces (CoVIS), October 2009.

Summary: In this paper, we presented design considerations for collaborative information visualization in multi-display environments and derived a system infrastructure for a multi-user information space using multiple discontinuous displays.
Contribution: main contributor to overall concept, in particular for interaction techniques, environment, and system design.
References: Section 10.3 (conceptual discussion on information manipulation and transfer).

- Manuela Waldner, Christian Pirchheim, Ernst Kruijff, Dieter Schmalstieg, **Automatic configuration of spatially consistent mouse pointer navigation in multi-display environments**, Proc. of the international conference on Intelligent user interfaces (IUI), pages 397–400, February 2010.
  Summary: In this paper, we describe spatially consistent mouse pointer navigation techniques in multi-display environments, which are automatically configured based on the spatially aware environment of Deskotheque.
  Contribution: concept and implementation of navigation techniques (except for world-in-miniature); main contributor of user study design, conduction, and evaluation.
  References: Section 5.3 (exemplary navigation techniques), Section 6.1 (comparison of navigation techniques).

- Manuela Waldner, Dieter Schmalstieg, **Experiences with Mouse Control in Multi-Display Environments**, Proc. of Workshop on coupled display visual interfaces (PPD), pages 6–10, May 2010.
  Summary: This paper reports on observational results and implications from an experiment comparing different mouse pointer navigation techniques in a multi-display environment.
  Contribution: see previous paper.
  References: Section 5.1 (cross-display navigation design space), Section 5.3 (exemplary navigation techniques), Section 6.1 (comparison of navigation techniques).

- Manuela Waldner, Werner Puff, Alexander Lex, Marc Streit, Dieter Schmalstieg, **Visual Links across Applications**, Proc. of Graphics Interface (GI), pages 129–136, May 2010.
  Best Student Paper Award
  Summary: This paper reports on the design and implementation of visual links across applications, visually connecting related items across application windows.
  Contributions: contributing to idea and concept (mainly motivated by a cross-display and multi-user scenario which was not covered in the initial publication); design and implementation of links routing, rendering in the window manager; "evaluation" of observational user study results.
  References: Section 7.2.1 (central application coordination), Section 8.1 (visual

links), Section 9.1 (exploratory evaluation of visual links).

- Manuela Waldner, Ernst Kruijff, Dieter Schmalstieg, **Bridging Gaps with Pointer Warping in Multi-Display Environments**, Proc. Nordic Conference on Human-Computer Interaction (NordiCHI), pages 813–816, October 2010.
  Summary: This paper describes an experiment comparing pointer warping to seamless mouse pointer navigation in multi-display environments.
  Contribution: idea and concept; main contributor of user study design, conduction, and evaluation.
  References: Section 6.2 (evaluation).

- Manuela Waldner, Dieter Schmalstieg, **Collaborative Information Linking: Bridging Knowledge Gaps between Users by Linking across Applications**, Proc. IEEE Pacific Visualization Symposium (PacificVis), pages 115–122, March 2011.
  Summary: Collaborative information linking, which visually connects information across private and shared application windows to bridge knowledge gaps between users, is presented.
  Contribution: idea and concept; implementation of multi-user extension for visual links; design and implementation of collaborative information linking interaction techniques; design, conduction, and evaluation of exploratory evaluation.
  References: Section 7.2.1 (multi-user extensions to central application coordination), Section 8.1.2 (concept and interaction techniques), Section 9.2 (exploratory evaluation).

- Manuela Waldner, Markus Steinberger, Raphael Grasset, Dieter Schmalstieg, **Importance-Driven Compositing Window Management**, Proc. Conference on Human Factors in Computing Systems (CHI), May 2011 (to appear as full paper).
  Honorable Mention Award
  Summary: Importance-driven compositing optimizes the spatial window layout for maximum visibility and interactivity of occluded content in combination with see-through windows. Emerging window manager functions to minimize information overlap and to improve access to occluded information are presented.
  Contribution: main contributor to idea and concept; main contributor of window manager integration (CPU-side), as well as cut-away compositing; main contributor of design and implementation of window management functions; design, conduction, and evaluation of user study.
  References: Section 7.2.2 (window importance maps), Section 7.4 (window layout, compositing, and implementation), Section 8.2 (uncovering window function), Section 8.3.2 (semi-automatic window layout, used for display-adaptive window management), Section 9.3 (experiment).

The following persons have to be explicitly mentioned as they contributed a significant amount of work to the outcome of the research prototypes presented in this thesis.

- **Christian Pirchheim** of Graz University of Technology was one of the co-workers of the Deskotheque system. His main responsibility was the distributed software infrastructure of Deskotheque (Section 3.2): network communication, the first version of input redirection and multi-pointer support, window migration, and plugins for the Beryl window manager. In addition, he designed and implemented the Coin3D-based version of the world-in-miniature switcher used for input redirection in the Deskotheque environment (Section 5.3.3). He was also the primary "system administrator" of software and hardware in the Deskotheque lab until 2008. As building the Deskotheque framework and physical environment was a matter of years and was conducted as teamwork, there have also been countless contributions in components of the framework not explicitly mentioned here.

- **Werner Puff**, who was temporarily part of Graz University of Technology, created the single-user management layer of the visual links infrastructure and implemented the application scenarios presented as use cases for the infrastructure (Section 7.2.1). Additionally, he supported the porting of Deskotheque to Ubuntu 10.04 and Compiz, respectively, where he was mainly responsible for the window manager plugins. For his master's thesis, he extended the Caleydo framework [142] for distributed usage, and he supported the integration of the Caleydo framework into the Deskotheque framework.

- **Markus Steinberger** of Graz University of Technology is responsible for the GPU-side implementation of importance-driven compositing: the integration and adaption of the saliency GLSL shaders by Mendez *et al.* [156] (Section 7.2.2), the OpenCL-based layout algorithm, and the main part of the compositing shaders (Section 7.4). He continued working on the concept of importance-driven compositing and facilitated the concept of importance maps for optimal routing of visual links. This follow-up work is not presented in this thesis.

- **Ernst Kruijff** of Graz University of Technology helped with the design and evaluation of mouse pointer navigation techniques for multi-display environments (Chapter 6).

- **Alexander Lex** and **Marc Streit** contributed to the concepts of a collaborative information workspace and visual links across applications (Section 8.1), in particular.

- **Erick Mendez** provided his GLSL-based implementation for saliency computation [156], used for the creation of window importance maps (Section 7.2.2).

- **Markus Murschitz** was a master student of Graz University of Technology and created a flexible, hierarchical structured light process for multi-display environments (Section 4.1.1). This structured light technique was used many times to reconstruct more or less complex environments. He also contributed valuable additions and bug fixes to the Deskotheque master and server components for the calibration process.

- **Ralph Wozelka** was a master student of Graz University of Technology and implemented a parameterization of reconstructed surfaces based on least-squares conformal maps [141], as suggested by Raskar *et al.*[190] (Section 4.1.4).

- **Mark Dokter** is a bachelor's student of Graz University of Technology and created Synergy+MPX – a combination of the mouse pointer sharing tool Synergy [212] and multi-pointer X [115] – for flexible multi-pointer sharing across multiple machines (Sections 2.3.2 and 3.2.2).

- **Joris Bayer** is a bachelor's student of Graz University of Technology and implemented a Compiz-based world-in-miniature [227] for window relocation and input redirection in multi-display environments using a single machine (mentioned in Section 5.4).

- **Albert Walzer** of Graz University of Technology created all videos for the papers and contributed some valuable ideas (*e.g.*, semi-automatic window coordination as window manager function, as described in Section 8.3.2).

- **Andreas Wurm** of Graz University of Technology was a great support in managing the Deskotheque laboratory. Also, **Albert Walzer** and **Christian Pirchheim**, already mentioned further above, invested considerable effort in building and maintaining the lab.

Chapter 2

Background and Related Work

The goal of this thesis is to design and develop interfaces and interaction techniques to facilitate visual information management in emerging display environments. Similar to the focus areas of this thesis, the discussion of related work will be separated into two categories: visual information management (Section 2.1) and display environments (Section 2.2). Finally, enabling technologies that facilitated the development of the research prototypes in this thesis will be presented (Section 2.3).

2.1 Visual Information Management

This related work section will discuss two areas of visual information management: general information presentation techniques, like multiple coordinated views, focus+context, magic lenses, and cue-based techniques (Section 2.1.1), and window management (Section 2.1.2). Subsequently, collaborative aspects of information management will be discussed (Section 2.1.3). The related work on visual information management will be finished with a discussion (Section 2.1.4).

2.1.1 Information Visualization and Presentation

Carpendale and Montagnese [54] divide *interfaces for visual access to information* into two components: *representation* and *presentation*. The representation component is responsible to map abstract data to a visual structure that can be displayed on the screen, such as maps, charts, or graphs. The presentation component is responsible to display the resulting image and to organize elements of interest. Interaction techniques, such as panning, zooming, or applying visual distortion, lead to changes in the presentation. The focus of this thesis lies on the second component: information presentation.

Information visualization is responsible for the visual encoding or representation of data. Usually, this involves massive amounts of data, and the research field addressed by an information visualization tool is mostly very specialized and narrow [103]. Card *et al.* [53] define visualization as

> the use of computer-supported, interactive, visual representations of data to amplify cognition.

With cognition, they refer to acquisition or use of knowledge. Examples of well-known information visualization systems are *Spotfire* [2], *prefuse* [103], or *Cerebral* [16]. As this thesis does not have information visualization as its primary focus, information representation techniques will not be discussed in more detail here. The interested reader should refer to basic literature on information visualization (*e.g.*, [53, 136]) for more details.

An important goal of information *presentation* techniques is to display large data sets. Limiting factors for the display of these data sets are usually given by the *screen real estate problem* [54] and by deficiencies of the human visual system or cognitive abilities to distinguish between important and unimportant information [59].

A popular method to overcome these limitations is to put more emphasis on the information currently in the user's *focus*, while compressing the *context* information to fit to the screen. Cockburn *et al.* [59] distinguish the following interface types for separating focus and context information:

- *Spatially separated* interfaces (overview+detail) show an overview and a detailed view synchronously.
- *Temporally separated* interfaces (zooming) use (de)magnifying to separate views temporarily, while keeping the views in place.
- *Seamlessly integrated* interfaces (focus+context) show focus and context views simultaneously – usually by employing geometric distortion.
- *Cue-based techniques* modify the rendering of focus objects or introduce proxies (*e.g.*, text labels).

The purpose of these techniques is not necessarily limited to compressing information to overcome limited space (or time) resources. For instance, spatially separated, coordinated views can also be useful for comparison of different datasets or to allow for more diversity by providing complementary details in each view [259].

In the following, we will discuss three interface categories for information in more detail: multiple coordinated views, seamlessly integrated focus and context techniques, and cue-based techniques. The fourth category mentioned above – namely zooming interfaces, like *Pad++* [26] – will not be discussed, as its relevance for this thesis is not directly given. In addition, the benefit of zooming interfaces for visual information management has only been demonstrated for small visual data sets [183]. For larger data sets, multiple windows introduced a lower demand on the visual working memory.

Multiple Coordinated Views

Multiple coordinated views employ two or more visualization views for data investigation. A wide-spread type of multiple coordinated views consists of exactly two views (dual-view

coordinated visualizations [62]), such as overview+detail, difference views, or world-in-miniatures [227].

In contrast to simple uncoordinated display in multiple separated application windows, an important aspect of multiple coordinated view applications is that information representations and operations on the views are coordinated [199]. North and Shneiderman [173] established a taxonomy of multiple window (or view) coordination, which distinguishes two dimensions: the basic user actions to be synchronized (*i.e.*, selecting items and navigating views) and whether the presented information is the same or different (but somehow related) in two views.

Many information visualization systems rely on multiple coordinated views, for instance *LinkWinds* [126], *Improvise* [261], or *Caleydo* [142], to name only very few examples. To overcome screen space limitations, individual views are sometimes perspectively distorted, like the free-floating view planes in *VisLink* [61] or the "bucket" arrangement in *Caleydo* [142] (Figure 2.1). Such view arrangements support information exploration by exploiting the user's spatial cognition, but introduce visual distortions to some views.

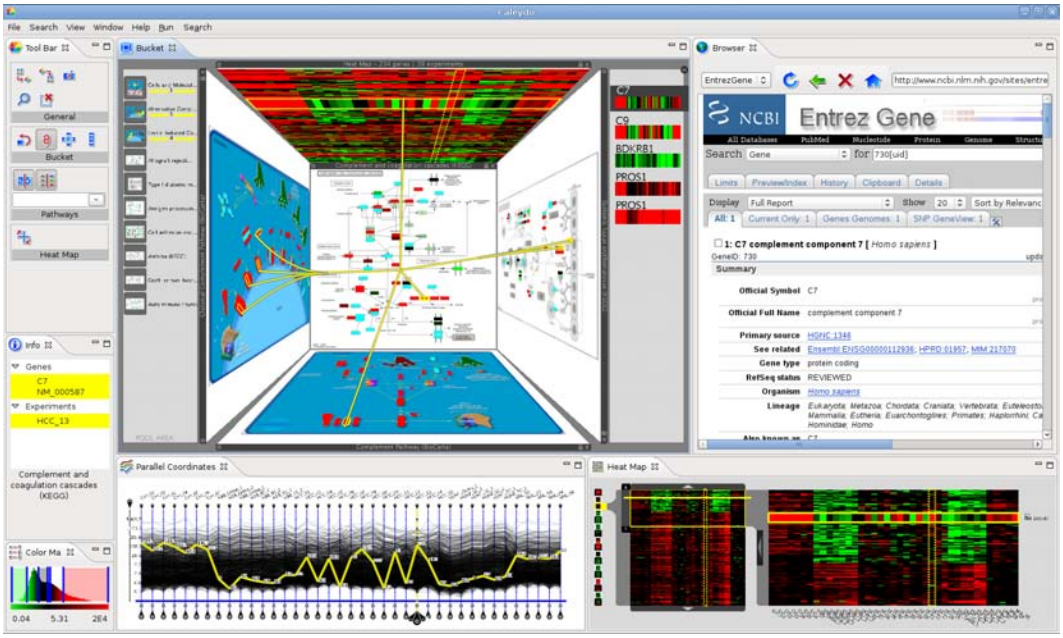


Figure 2.1: Caleydo [142] integrates multiple coordinated views in simple side-by-side or tabbed arrangements and in the "bucket". Visual links connect related elements in the bucket views.

North and Shneiderman [174] pointed out the strength of coordinated views for tasks involving analysis of overview and detailed information on demand. Plumlee and Ware [183] showed that multiple windows are more efficient than zoomable interfaces if the number of items to be investigated is high and, as a consequence, the demands on visual memory are high. Wang Baldonado *et al.* [259] set up rules for multiple view usage and recommend the employment when the information can be represented in diverse ways (like

overview+detail), if multiple views illustrate correlations, or if complex data can be split into smaller, easily manageable chunks. They also pointed out the importance of making the relationships among multiple views apparent to the user.

Typically, multiple coordinated views are embedded into a specialized information visualization framework. The framework is responsible for the coordination of the representation and user operations. This is a main distinguishing aspect between multiple coordinated views embedded into a visualization framework and multiple sources of information displayed by different applications in separate windows on the desktop: Operating systems treat individual application windows independently without coordinating their content, their visual representations, or visualizing correlations across multiple windows. Stone *et al.* [228] address this issue theoretically when discussing software architectures for dynamic queries across applications: A standard language for a rich object description for applications with specific semantics is required. In addition, modifications to low-level graphics libraries would be necessary to alter the appearance of application content in a unified way.

As a first step towards this vision, *Snap-Together Visualization* [174] treats conventional application windows as multiple coordinated views by "snapping together" visualization views by different applications. Snapped applications provide mechanisms such as synchronized highlighting and scrolling. Highlighting is accomplished by the specific applications. Thus, no visually consistent synchronized highlighting is available.

Focus+Context and Magic Lenses

In information visualization, focus+context techniques are well-established to allow for a large overview integrated with details without spatial separation. A focus area – which is usually movable – is surrounded by a context area, which is often compressed or distorted (*e.g.*, *Fisheye views* [85], the *Perspective Wall* [147], or the *Document Lens* [203]).

Instead of distorting views on a small-scale display, others facilitated special display arrangements to seamlessly integrate high-resolution focus information with low-resolution context. The *focus+context screen* [21] surrounds a high-resolution monitor with a low-resolution projection. Similarly, *Escritoire* [8] superimposes a low-resolution tabletop projection with a high-resolution inset. In contrast to these static insets, *PixelFlex* [266] and *foveal inset* [224] use a pan-tilt projector unit to achieve a movable, high-resolution focus area within a lower-resolution large-scale projected display. A comparative evaluation showed that using a focus+context screen increases performance and reduces error rates compared to zoom and pan and overview+detail interfaces on single and dual monitor settings, respectively [20].

In contrast to the above mentioned focus+context techniques, magic lenses do not have a continuous transition between a focus and a context region, but rather apply a hard boundary between the conventional view and a filtered representation. Magic lenses were initially described as movable visual filters that change the representation of the underlying

data [38]. They can be used as virtual magnifying glasses, to reveal hidden information by showing an in-place complimentary view [228], or to filter the data representation by dynamic queries [77]. As magic lenses do not have a continuous transition between focus and context, Cockburn *et al.* [59] do not consider magic lenses as seamlessly integrated views. They argue that magic lenses actually apply a spatial separation on the z-axis and therefore assign magic lenses to spatially separated interfaces, like overview+detail or multiple coordinated views.

The operation of the magic lens was initially designed for the non-dominant hand, while controlling the cursor with the dominant hand [38]. To ease interaction, many investigations concern the embodiment of magic lenses through physical objects and tangible interfaces. Ullmer and Ishii [247] distinguish active and passive tangible lenses for their *metaDESK*: Active lenses are self-contained, tracked display devices. In the metaDESK environment [247], the active lens was represented as an arm-mounted flat-panel monitor, which shows 3D geographical information augmenting the 2D map displayed on the table. In *Ubiquitous Graphics* [210], the user can get a more detailed view of a projected map by holding a tablet PC in front of the projected display. While this basic functionality is similar to a dynamic focus+context screen, the user can use the tablet PC also as input device to add annotations or virtual objects to the map. The high-resolution display and the possibility to provide input, if using a PDA, tablet PC, or cell phone, are advantageous features of active magic lenses. However, active magic lenses have the disadvantage that they are more expensive and fragile, compared to passive lenses.

Passive lenses are tracked props without display capabilities, which alter the appearance of a larger scale display within its boundaries. For the metaDESK [247], the passive lens is a transparent item which modifies the view of a map displayed on the tabletop surface. Spindler *et al.* [222] use non-transparent tracked cardboards (*Paper Lenses*), which are augmented by a front projection system, in combination with a rear-projected tabletop display. They introduced several tangible magic lens interaction techniques, such as free exploration of a volumetric space, as well as exploration of layered, zoomable, and temporal information spaces. For information visualization, they use tangible views for multiple view visualization and lens techniques [223].

Cue-Based Techniques

Cockburn *et al.* [59] established two categories of cue-based techniques: visual cues to highlight the focus objects and cues for visualizing off-screen locations.

Selections in individual views are traditionally expressed through conventional highlighting or color-coding. As an example, refer to the bottom views in Figure 2.1: the selected data set is color-coded in yellow in the parallel coordinates plot, framed in the heatmap, and the label of the data set is shown in the info bar on the left. As the views are coordinated, a user selection in one view is propagated to all the other views – an interaction technique which is usually referred to as *brushing* [154] in information

visualization.

A few systems use more expressive highlighting mechanisms to show the relationship of selected items across multiple views. *Spiral calendar* [148] indicates the relationships among multiple calendar hierarchies through half-transparent connections. Shneiderman and Aris [218] link categories of network visualizations to another. To avoid visual clutter, the user can filter the number of links by dynamic queries. In *VisLink*, Collins and Carpendale [61] visualize the relationships of heterogeneous 2D visualizations arranged in 3D space through multiple edge connections. They argue that these edge connections help to reveal relationships, patterns, and connections between views. Lex *et al.* [142] use visual links to show dependencies between 2D pathways (models of biological processes) and gene activities in their "bucket" (*cf.*, Figure 2.1).

Off-screen visualizations should help the user to discover – or even navigate to – information located outside the display region. This is a very common problem for visualizations on small-scale displays, such as handheld devices, where the amount of available screen space is very low. *Halos* [23] were developed for map interfaces on handheld devices. Halos are circles rendered around off-screen locations. The size of the circles is chosen so a small portion of the ring is visible at the display boundary. Thus, the location of the visible ring portion and its curvature inform the user about the location of the off-screen item and its approximate distance. *City Lights* [269] are 2D objects arranged along window boundaries, which indicate the existence, size, distance, and other properties of interest of clipped information. Irani *et al.* [118] combined Halos with a teleportation mechanism called *hop* to allow for easy recognition and navigation to off-screen objects.

2.1.2 Window Management

Card *et al.* [51] define a window as

> a particular view of some data object in the computer.

Windows can contain various types of visual information, such as text, images, charts, or videos [162].

Having multiple windows allows the user to interact with multiple sources of information simultaneously. A window manager is responsible to separate different contexts physically onto different parts of one or more display screens [162]. However, it is not responsible for the representation of the information contained within its boundaries. Card and Henderson [52] therefore use the term *placements* as description of the visual presentation of the window on the screen. It consists of a reference to a window and its content, a location, shape, and presentation attributes, like whether the window border has drop shadows. The advantage of this approach is that the application's visual output and the mapping on the display is conceptually separated. However, this notation is no longer used in today's window managers.

Card *et al.* [51] distinguish four types of window system designs:

1. simple TTY ("teletypewriter") windows, with single-direction scrolling and command input on the bottom of the screen,
2. time-multiplexed windows, where one window is visible at a time and new content can be revealed by scrolling or flipping between frames consecutively,
3. space-multiplexed windows, where multiple windows are shown concurrently on the screen, either separated along one or two dimensions, or by 2.5D arrangements, where windows can partially overlap, and
4. non-homogeneous windows, where the level of detail within a window or across multiple windows can change by zooming or fisheye techniques.

Space-multiplexing is the most wide-spread design of today's window managers – except for very small handheld devices, where time-multiplexing is more common. Space-multiplexed windows can be roughly divided into two categories [162]:

1. covered (or overlapping) windows and
2. tiled windows.

In the following, there will be a presentation of overlapping windows (in particular with respect to window switching) and tiled window managers, as well as hybrid techniques. In addition, depth-multiplexed windows and other window management approaches will be shortly discussed.

Overlapping Windows

Overlapping windows are the standard approach to modern window management. They correspond to the category of 2.5D space-multiplexing introduced by Card *et al.* [51], where windows have an (x, y) position on the two-dimensional screen space and a unique z-value, defining its depth order in the window stack. The z-value is usually assigned by its recency of use, leaving recently used windows on top of the stack and windows unused for a long period of time buried on the bottom.

One of the first computing systems using overlapping "windows" was the *Smalltalk* system by Xerox PARC [89] (1979). In the Smalltalk system, *views* had similar properties as our current windows [89]:

> The display screen contains one or more rectangular areas called views. The views are displayed on a gray background and may overlap. Each view has a title shown at its upper left corner.

The window concept has later been adapted by *Xerox Star* [132] (1981), Apple's *Lisa* (early 1980s) and *Macintosh* (1984), and Microsoft's *Windows* (1983).

To bring an occluded, or partially occluded, window to the front, window managers provide window lists, sequential window switching techniques (such as the temporal Alt+Tab

sequence), or space-filling window layouts (such as Apple's Exposé*). Gaylin [87] observed that the most frequent activity in early window managers was cycling through windows. Moving and resizing windows was rather conducted at the beginning of the interaction session (during the *task environment setup* activity [135]), and the thereby created window layout remained largely unchanged during the session. However, cycling through windows using Alt+Tab window switching has been described as tedious by many users [91].

Therefore, when users have sufficient screen space, they "carefully coordinate" their windows to keep a small portion of occluded windows visible [113]. This way, they can directly select a partially obscured window by clicking onto its visible area and thereby circumvent explicit window switching, such as Alt+Tab. Bi and Balakrishnan [32] observed that, especially when working with very large displays, users spend a significant amount of time resizing and moving windows, as opposed to small-scale displays, where windows are often simply maximized to select. But despite this increased window management overhead, users overwhelmingly preferred working with a large display.

To ease the switching between layers of windows, more advanced switching interfaces have been created. For instance, Faure *et al.* [75] group non-overlapping windows to common z-layers and provide a crossing-based interface to quickly select another window layer. Tak *et al.* [236] found out that spatial consistency of window switching interfaces is more important than recency-based consistency. Their space-filling tree-map [131] window switching interface therefore considers the spatial history and frequency of use when laying out the windows on the screen [235].

Others group windows by assigning them to different tasks. Switching between tasks brings a new set of windows to the screen. One of the first examples of a task management system was *Rooms* [70]. Each virtual room contains a set of windows, and virtual doors connect these rooms. The principle was adopted by *virtual desktops*, which is a standard feature of many Linux desktop environments. Another prominent example is *Scalable Fabric* [201], where the screen is partitioned into a central focus area and a peripheral context area. In the focus area, windows are treated as conventional overlapping windows. By dragging windows into the periphery, they are scaled down. Users can drop them onto existing piles and thereby create tasks consisting of multiple windows. Clicking on such a task restores all contained windows to the focus region. Similarly, *Task Gallery* [202] assigns windows to tasks and displays them on different surfaces in a virtual room. The primary task is displayed on the "stage", where windows are arranged in a conventional overlapping fashion. As the user "backs up" from the stage, the peripheral walls are shown and she can navigate to adjacent rooms.

In these examples, the user has to assign application windows to the tasks manually. Techniques to assign windows to tasks automatically have also been explored, for instance by analyzing the switching history and comparing window titles [175].

*`http://www.apple.com/macosx/what-is-macosx/expose.html`

Tiled Windows

Tiled window managers arrange windows side-by-side without overlap. Simple tiled window managers arrange windows in one dimension only, while more sophisticated ones support 2D grids, constraint-based tiling [60], or hierarchical setups [135]. Bly and Rosenberg [43] compared tiled and overlapping windows in a user study. Their results showed that users performed a decreased amount of window management operations with tiled windows and that handling overlapping windows required a higher level of training. However, most users preferred overlapping windows to tiled windows.

Tiled window managers have the advantage that the user does not have to resize and position windows manually to fill the entire screen space. It assures that no space is left empty and leads to a decreased amount of manual window management operations. However, especially on small-scale displays, a tiled window manager has to resize or even close certain windows to avoid overlap. Automatic down-sizing introduces the risk of clipping important content at the window's boundary. Miah and Alty [157] overcome this issue for document editors: when scaled down, they visually highlight the most important document keywords to support the users in window recognition. For such an approach, the window manager requires access to the window's application content.

Hybrid Tiled-Overlapping Windows

Strictly tiled windows do not play an important role in modern window managers any more. Tiling is merely a semi-automatic feature in many window managers today, where overlapping windows are automatically positioned and resized to allow for quick side-by-side comparison without tedious manual arrangement. For instance, in Windows XP, users could select windows to be tiled vertically or horizontally in the taskbar. In Windows 7, the user can drop windows at the window boundaries to maximize them to the half of the screen. In the Grid plug-in[†] of the Compiz window manager (*cf.*, Section 2.3.1), users can employ keyboard shortcuts to move and resize windows into a regular grid. In all these examples, windows are placed and resized without considering their content.

To reduce the risk of cropping important elements of application windows, *Snipped Windows* [114], and *Window Clips* [155] only show the most relevant regions of context windows (*i.e.*, windows which are providing background information and do not currently hold the user's input focus). These cropped window regions are arranged at the screen periphery – either as conventional overlapping windows [114] or in a sequential list [155]. Thereby, the majority of the screen space can be left to the focus window(s) while maintaining important elements of peripheral windows. Similarly, *WinCuts* [238] are small, freely movable windows containing a copy of dedicated screen content for easier visual comparison and manual arrangement of relevant content. *User interface façades* [232] allow for re-configuration of certain user interface elements (*e.g.*, buttons) or other arbitrary

[†]`http://wiki.compiz.org/Plugins/Grid`

window regions into a new parent window – the façade. However, in these systems, the user has to define the relevant regions manually, which can be a tedious task.

Other hybrid techniques between strict tiling and overlapping windows are systems that re-locate or resize windows semi-automatically. One example is overlap-avoiding dragging [27], which relocates windows after a dragging operation to avoid overlaps. The authors presented two possibilities of overlap-avoidance: either the window being dragged snaps to the closest empty location after releasing or the windows being covered by the released windows are relocated. The optimal window layout is calculated by consulting a dynamic representation of 2D space, consisting of *full-space rectangles* (*i.e.*, application windows or other user interface elements) and *empty space* (*i.e.*, the desktop background). Badros *et al.* [10] presented a constraint-enabled window manager, where users can manually define a set of constraints on windows, which are evaluated to create an optimal window layout. For instance, a user can bind a window to another window's edge or restrict a window's size or distance to a screen edge. In both examples, a sufficiently large screen is required to accommodate for all the non-overlapping windows.

Other techniques change the window layout only temporarily in certain situations. Chapuis and Roussel [56] fold or roll partially occluding windows away if the user selects text in an occluded window for copy-and-paste. Occlusion-aware interfaces [251] relocate user interface elements (or windows) when a physical occlusion on a pen-based system has been detected.

Except for manually cropped windows [114, 155, 238], all of the above mentioned window management techniques treat windows as simple rectangular shapes – irrespective of their content. This does not leave much room for (semi-)automatic window layout if the screen is cluttered with a large number of windows.

Depth-Multiplexing

As an additional window management category to space-multiplexed windows and time-multiplexed windows introduced by Card *et al.* [51], Harrison *et al.* [98] include depth-multiplexed windows. They are based on the idea of see-through user interfaces, initially introduced by Bier *et al.* [38]. Depth-multiplexed windows use transparency to reveal the content of the obscured windows in an overlapping window interface. However, experiments of transparent dialogs over 3D objects [99] and images [22] have shown that simple alpha blending either causes readability problems with the dialog items or decreases the perception of the background image – depending on the chosen alpha level. Outlines of text and icons [98] or *multiblending* [22] have been shown to improve recognizability of blended content.

In a window management environment, Zanella and Greenberg [268] apply transparency only if window regions owned by different collaborators overlap. Ishak and Feiner [123] apply free-space transparency only to "unimportant" window regions. They define unimportant regions as those with white pixels. This guarantees that "important" con-

tent in top-level windows is preserved, while unimportant regions can be used to reveal obscured content.

None of these techniques guarantees that important content in occluded windows or views are actually revealed. If important content in the occluded window is located underneath the important window regions of the overlay item, the user has to manually re-arrange windows or menus to access hidden content.

Other Window Management Approaches

This section shortly discusses window management techniques that do not apply any spatial or task-based organization, such as time-based interfaces or physics-based interfaces.

Time-based interfaces allow users to manage windows based on a history. *Time-machine computing* [194] is a history approach to file management. *WindowScape* [242] extends this concept to window managers: each time the user re-arranges a window, a snapshot of the environment is created, which the user can re-instantiate at any time. *Mnemonic Rendering* [31] buffers individual pixels in (partially) occluded windows and replays the changes of a pixel in high speed, if it becomes visible again. The aim of this technique is to help the user perceive and understand invisible content changes.

Other window managers aim at physical realism. Beaudouin-Lafon [25] introduced *window stacks*, a paper-like presentation technique, where overlapping windows are rotated and peeled back to reveal regions of underlying windows. *BumpTop* [1] uses physics simulations to organize desktop icons and windows on a virtual desk. Users can create piles, leave through items, or fan out their desktop items on the virtual desk surface. *Paper Windows* [109] is a tangible window manager, where individual windows are attached to deformable, rectangular objects, similar to physical paper. Different physical window manager operations were introduced.

2.1.3 Collaborative Information Management

Almost all of the information management techniques presented in this section were designed for individuals. However, deriving knowledge from a wealth of complex information sources and making decisions based on this knowledge is a task where teams are superior to the individual [104].

A typical collaborative work style observed when dealing with problem solving in teams is *mixed-focus collaboration* [94]. These mixed-focus situations comprise times of close collaboration (tight coupling), as well as times when people work individually (loose coupling) [208]. In a co-located setting, there is frequent switching between those work styles [74, 94, 122, 240]. Basic operations of teamwork have been summarized in the *mechanics of collaboration* [95, 179]. The mechanics are divided into the general categories *communication* and *coordination*, and describe operations team workers have to accomplish to complete a collaborative task. Examples are spoken and gestural messages, basic awareness, obtaining, reserving, and protecting resources.

In the field of computer-supported cooperative work (CSCW), numerous *groupware* applications have been presented for collaborative information search, browsing, and analysis. Groupware is usually categorized along the time and space dimensions (*cf.*, [11]): it can support synchronous or ansynchronous communication (time), as well as one meeting site or multiple meeting sites (space). The collaborative aspects discussed in this thesis will concentrate on synchronous, co-located collaboration, *i.e.*, a single meeting site where multiple collaborators are present at the same time.

Examples for synchronous, co-located groupware for information management range from collaborative picture galleries [158], web search [3, 246], tree comparisons [120], text analysis [121], and graph visualization [119], to more general data visualization in multiple coordinated views [243]. An important aspect of collaborative information management is that experts from different fields may prefer different information representations based on their field of expertise, subjective preferences, and role within the examination process [104]. Wang Baldonado *et al.* [259] therefore state in their "rule of diversity" that multiple coordinated views should be employed if users from different fields collaborate. On the other hand, Heer and Agrawala [102] outline the importance of sharing the same visual environment to establish "common ground". Convertino *et al.* [63] therefore proposed a single team view and role-specific views for each team member to ease the group analysis task. In *Lark* [243], users can "branch" coordinated views from a shared visualization pipeline, connected to a common data source. In contrast, Tang *et al.* [240] as well as Forlines and Shen [81] demonstrated systems providing each user with tools for personalized filtering of a single shared view.

In most of the above listed examples, collaboration support is guaranteed by using special-purpose collaboration-aware software tools. Mostly, these groupware applications are self-contained developments. In contrast, Isenberg *et al.* [119] showed that existing visualization software can be "retro-fitted" for multi-user interaction – for instance by rendering multiple color-coded mouse cursors within the application. Forlines *et al.* [79] presented a wrapper for Google Earth to support multi-user input, viewport synchronization across multiple instances, and annotations, without changing the application's core implementation. Yet, applications themselves must be modified for collaboration-awareness. As a consequence, these applications cannot easily be combined with other groupware applications for a fluid information exploration process.

An early vision of the field of CSCW was to use existing single-user applications in collaboration-aware windowing systems instead of special-purpose groupware applications [140]. First steps towards this vision have been undertaken by providing multi-input support in conventional operating systems (*e.g.*, *multi-cursor window manager* [254] or *multi-pointer X* [115] – *cf.*, Section 2.2.4). These solutions allow multiple input devices to control distinct application windows concurrently. However, they cannot coordinate multi-user input on a single application window if the application itself is not multi-pointer aware.

Window management techniques tailored to collaborative information management

are scarce. The primary objective of window management techniques specifically designed for multi-user interaction is to coordinate multi-user activities and to reduce interference. Zanella and Greenberg aim to reduce interference in window interfaces by dynamically applying transparencies if an interface component raised by one user occludes active elements by another user [268]. Shoemaker and Inkpen eliminate interference using *single-display privacyware* [219] which separates the output channels for two users operating on the same display. Therefore, they can super-impose personalized information, which cannot be seen by the other user. Morris *et al.* [159] presented a set of automated multi-user coordination policies for resolving conflicts on tabletop displays. Although the coordination policies were discussed conceptually for arbitrary documents, they could be implemented by a window manager. Hutterer and Thomas [115] allow users to lock individual application windows to prevent other users from interacting with their content. On the public display of *Dynamo* [125], users can define private screen regions inaccessible to other users. Tsandilas and Balakrishnan [244] categorize techniques for reducing spatial interference on single-display groupware into three categories: *shared screen* but individually locked elements, *split screen* with private screen regions per user, and *layers* on the z-axis, where interference is reduced by transparency. Results of an experiment showed that the last category achieved the best performance.

Other collaborative window manager or display space management techniques help multiple users to arrange their shared information for easier comparison. In *WeSpace* [264], users can lay out remote desktop windows on a shared display wall using a dedicated layout manager or *LivOlay* [127], which overlays spatially registered windows for easier comparison. *WinCuts* [238] allows users to "cut out" regions of windows and to freely move these regions like separate, small windows. In this way, relevant information from multiple sources and users can be arranged more efficiently.

2.1.4 Discussion

To summarize the related work and background of visual information management, a brief discussion will outline the most important aspects with respect to this thesis: the conceptual differences between multiple coordinated views and application windows, window management beyond the box, and window management beyond the individual.

Multiple Coordinated Views vs. Application Windows

Multiple coordinated views and window managers are conceptually very similar: They show information in separate areas of the screen and the user can manage the spatial arrangement of these information containers. In both areas, information presentation techniques have been introduced to minimize the screen real-estate problem. For instance, the *Perspective Wall* [147] was an archetype for the window and task management system *Task Gallery* [202], and similar to the windows attached to the virtual walls in the gallery,

views are attached to the sides of the bucket in the multiple coordinated view system *Caleydo* [142] (Figure 2.1).

In fact, considering the framework for a unifying presentation space of Carpendale and Montagnese [54], the input for the presentation space is a texture-mapped surface. Both, self-contained multiple coordinated view systems and window managers can handle their individual views or windows, respectively, as texture-mapped surfaces. Until recently, treating windows in such a generalized fashion was rather tedious (*cf.*, [249]). However, modern 3D compositing window managers (*cf.*, Section 2.3.1) allow user interface designers to access windows in such unified way: as textured quads in a 3D environment. Presentation operations and interaction techniques can either be applied on selective windows or views individually, or on the resulting desktop image or information space as a whole. In this way, image-based presentation and interaction techniques, such as geometric deformations (*e.g.*, fish-eye distortions [85]), appearance modifications (*e.g.*, transparencies), layout optimizations of individual elements, or visual cues rendered on top of the output image, can be easily applied within the window manager – just like within a multiple coordinated view system.

A limitation of window managers, compared to self-contained multiple coordinated view systems, is the inability to access semantic information from the individual applications. Cue-based focus and context techniques (*e.g.*, [61]) or dynamic queries (*e.g.*, [228]) cannot be implemented without further semantic information from the applications. A system infrastructure like *Snap-Together Visualization* [174] can serve as a foundation for semantic information exchange.

Visual links across applications presented in this thesis (Section 8.1) lend ideas from multiple coordinated view systems for advanced window management. A simple application coordination mechanism (similar to *Snap* [174]) manages user selections in different application windows (Section 7.2.1). The focus of this work lies in the visual coordination: an explicit and unified highlighting mechanism visually connects related items contained in different application windows on the desktop.

Window Management beyond the Box

With the access to window textures through the use of compositing window managers, we cannot only apply geometric or image-based transformations, like fisheyes [85]. We can also access and manipulate the visual window content by analyzing and modifying their textures. But window management techniques operating on a pixel level are still scarce: some transparency techniques (*multiblending* [22] and *free-space transparency* [123]) and *mnemonic rendering* [31] are notable exceptions. Classic window manager tasks, such as window switching interfaces or window layout techniques do not consider the visual window content for determining important or unimportant window elements. Thereby, valuable screen space is wasted for potentially unimportant information, while important elements may be hidden behind application windows or cropped by the window boundaries.

In this thesis, examples for window management beyond the box to facilitate the discovery of important information will be presented. Importance-driven compositing window management (Section 7.4) is a novel window management approach explicitly exploiting the windows' textures for an optimized spatial window layout. It analyzes the visual window content and uses visual saliency [124] to determine perceptually important window regions (Section 7.2.2). The window layout engine then calculates a spatial window arrangement that aims to preserve these important window regions. See-through compositing (similar to free-space transparency [123]) is employed to reveal occluded content. To illustrate the usefulness of the technique, we present a new window management technique to improve the access to occluded content (*uncovering windows* in Section 8.2).

Window Management beyond the Single User

Today's window managers are designed for a single user interacting with one input device. Even if the underlying operating system provides multiple input streams (for instance by using multi-pointer X [115]), window managers have to fulfill two important requirements to successfully support multiple concurrently interacting users on the same display: coordinating multi-user input by applying access restrictions and coordinating the visual information associated with different users. As some authors consider system-imposed access coordination as too restrictive and argue that social protocols are sufficient to negotiate access control (*e.g.*, [72, 90]), the second requirement can be considered as the more relevant one.

Existing visual coordination techniques on window manager level, such as dynamic transparencies of overlapping windows [268], channel separation techniques [219], or semi-automatic layout of collaborators' remote desktop windows [264] reduce interference but do not support users in other mechanics of collaboration [95], such as workspace awareness or gestural referencing. Systems supporting a rich set of communication and coordination mechanics are usually embedded into special-purpose applications. This way, user-specific selections can be synchronized or highlighted across multiple views, findings can be collected and prepared in a consistent manner, and shared views can be individually filtered.

In this thesis, multi-user coordination techniques requiring no or very limited modifications of legacy applications on window manager level are presented. Collaborative information linking facilitates user-specific visual links across applications to afford visual coordination. Multiple single-user applications can contribute to a shared information analysis task by providing user-specific visual feedback and a shared visual context on public application windows (Section 8.1). Optical magic lenses are low-cost, tracking-less magic lenses exploiting properties of polarized light to switch between two information layers – completely transparent to the application. In this way, users can filter a shared view with a passive magic lens without changing the application's visual output (Section 8.4).

2.2 Display Environments

There are numerous reasons why users prefer the usage of large displays or multiple displays to a single small-scale display. In the following, a few of these reasons are listed:

Overcoming the screen real estate problem. The most obvious reason why users prefer larger displays with a higher number of pixels is the fact that more information can be displayed simultaneously if more pixels are available. Even though a larger amount of physical navigation may be required to explore a large, high-resolution information space, users can perform an information analysis task on a large display faster than on a small display with panning and zooming [14]. In particular, if users have to perform a cognitively demanding task, having a large display increases performance [69] and subjective satisfaction [4].

Partitioning the information space. If a lot of display space is available, users do not necessarily clutter all the available space with focus information. They rather explicitly facilitate physical discontinuities for partitioning their information spaces. A primary display is employed to show focus information, while a secondary display shows supportive applications [91]. On a single large-scale display, users implicitly create spaces for storage [215], windows or icons acting as reminders [113], clusters and piles of windows [4], as well as focus and surrounding context regions [32].

Enabling multi-user interaction. A large, high-resolution workspace is required if a lot of information in personalized visual encodings is displayed concurrently [120]. It is also crucial for users to establish their personal territories and to coordinate their activities [245]. However, multi-user interaction is not only restricted by the limited visual output on a small-scale display, but also by the physical space in front of the display. With large-scale displays, multiple users can view and operate a shared information space simultaneously without major distractions. As different display form factors enable different collaboration styles (*cf.*, [74, 117, 152, 204]), multiple displays of various form factors are often combined into a common interaction space.

However, large displays and multi-display environments also lead to new challenges in terms of user interface and interaction design. Robertson *et al.* [200] and Czerwinski *et al.* [68] list a number of usability issues arising from large monitor interaction, such as losing track of the cursor, accessing distant information, window management problems (*e.g.*, popping up windows at unexpected locations), multi-tasking problems, and insufficient support for peripheral activities. Swaminathan and Sato [234] list the difficulty of locating items of interest on a display scattered with information as one of the most important challenges. In collaborative settings, typical problems on large, shared displays are the inability to track other users' cursors [264] and gesturing to distant display locations [101, 119, 204].

In multi-display environments, having physical discontinuities across multiple displays, such as size-resolution mismatches and large physical gaps, leads to additional problems. These discontinuities cause navigation difficulties [18], segmentation of visual information [146], and visual inconsistencies of displayed information [169]. For collaborative information management, studies have shown that coordination of access to shared resources is decreased [258] and workspace awareness is generally lower [152, 204] if activities are distributed across multiple displays.

Another aspect of large displays and multi-display environments is the hardware and software infrastructure. Many user interface prototypes for such display environments rely on dummy applications or special-purpose tools. Large-scale displays are often created by careful manual adjustment of the individual projectors, or by combining multiple monitors to a large matrix. The latter approach introduces physical bezels that interfere with a seamless information display.

In the following, evidence on the effects of display factors on interaction and collaboration collected in related research will be presented (Section 2.2.1). Then, the design and creation of large-scale displays will be discussed (Section 2.2.2), followed by a presentation of multi-display environments (Section 2.2.3). The literature review will be concluded by a section on middleware (Section 2.2.4) and a discussion (Section 2.2.5).

2.2.1 Effects of Display Factors on Interaction and Collaboration

With the wide-spread adoption of projectors, new display form factors and display arrangements have emerged. In contrast to the rise of very small, handheld devices, projected displays are usually stationary, larger than conventional monitors, and located at a distance from the user. In addition, the low cost of display hardware made a large number of high-resolution monitors available to the end user, often employed in the same environment as projected displays.

Several researchers have demonstrated a productivity benefit for users when working on large, vertical displays. Czerwinski *et al.* [69] pointed out the strengths of large displays for cognitively demanding tasks. Ball and North [14] observed that tiled, high-resolution displays increased performance of users in a visualization task, compared to single monitors where users had to pan and zoom to access all information. Bi and Balakrishnan [32] demonstrated that users appreciated tiled, high-resolution projected displays with seamless imagery, although window management support proved to be sub-optimal for such a display configuration. Andrews *et al.* [4] observed the usage of large display space as external memory to support a sensemaking task. On the other hand, Forlines *et al.* [82] showed that individuals had a better performance in a visual search task when using only a single monitor. Adding complimentary views (identical but rotated views) to multiple displays lead to a performance decrease while not improving the error rate.

In collaborative environments, display form factors diverging from the conventional vertical screen are increasingly common. The physical affordances of a display – such

as its accessibility, visibility, shareability, and directness of interaction [204], as well as its location within the environment [177] – implicitly suggest a function and role. There have been numerous observations of co-located collaboration around displays of various form factors. For instance, it has been demonstrated that mixed-focus situations (*cf.*, Section 2.1.3) are particularly well supported by tabletop displays [74, 215]. On the other hand, tabletop displays suffer from representation ambiguities caused by the seating arrangements of the users [263, 265].

Conversely, wall displays provide a convenient shared view for all participants. Researchers suggested to employ wall displays for comparison tasks and presentations [74], group information [152], overviews [117], and peripheral information [143]. When interacting with vertical displays, a more asymmetrical group interaction emerges compared to tabletop interaction [74, 152, 204]: Often, a single user takes control while the others act as passive observers. However, others have shown that asymmetry can be overcome by facilitating multiple concurrent input devices and sufficient space to establish personal territories for uninterrupted work [41, 245, 246]. Another distinguishing aspect of wall displays concerns the distance of the users: often, users are located farther away from the display compared to tabletop displays. Observations have shown that users had more problems resolving physical gestures towards the display when located far away from the screen [101, 119, 204].

Several researchers recommended introducing smaller personal display spaces for individual work [74, 152, 214, 240]. Observations showed that multiple displays caused a better coordination of distributed work [204] and a more equitable task workload [117]. However, when users do not have a shared (public) view, it becomes difficult to maintain awareness of each other's actions [94, 152, 204] and to discuss and share ideas [117]. Even when sharing a common view on a large public display, an increased amount of coordination problems compared to *single-display groupware* [226] has been observed, due to decreased awareness [258].

Multiple displays are not only advantageous for multi-user interaction. Grudin [91] and Ringel [198] found that users explicitly facilitated the physical separation afforded by multiple monitors to divide their tasks into focus activities and peripheral activities, as well as for document comparisons. However, Hutchings and Stasko [113] pointed out deficiencies of current window managers with respect to multi-display settings.

2.2.2 Large Displays

Large displays with a high resolution are often created by combining multiple monitors to a large multi-monitor matrix (*e.g.*, [13, 14]). However, such a multi-monitor matrix suffers from a large number of bezels. Therefore, another approach is to use multiple projectors to create a single seamless image. The discussion of large displays in this thesis will focus on the creation of irregular and tiled *projected* displays, the content creation for such displays, and a selection of interaction techniques for large displays.

Irregular and Tiled Projected Displays

Manually adjusting multiple projectors to create seamless imagery can be a tedious task. Especially when building tiled displays in conventional office environments, the limited wall and table space often restricts the usage of projected displays. The resulting images may be distorted due to oblique projection angles. Overlap regions of non-uniform brightness may result from improper projector alignments.

There are different solutions to calibrate projector-based displays to compensate for keystoning effects and to provide uniform tiled displays automatically. Usually, one or more cameras are employed to detect the display surface using structured light patterns. There are different strategies of coded structured light, such as time-multiplexed codes (*e.g.*, binary gray codes), color encoding, or simple checkerboards [209]. The special class of imperceptible structured light aims to make structured light patterns invisible to human observers. In the *Office of the Future* [192], cameras synchronized with projectors capture patterns projected at a single frame. In the subsequent frame, the projector displays the complimentary image, so the light is effectively canceled for the human observer. More recently, Cotting *et al.* [64] exploited mirror sequences of DLP projectors to embed binary patterns into full-color images. Input color values are minimally adjusted to achieve the desired mirror position (on or off) at a dedicated time frame, when the camera captures the display. Another recent approach is to observe natural features in the projected imagery in real-time for portable projectors [133]. With this approach, no specialized hardware or hardware modification is required.

Based on the obtained camera images from the structured light process, two calibration processes can be conducted [48]: geometric registration and photometric correction. In this thesis, the photometric component will be neglected. An overview of existing techniques can be found in [48].

A common approach to describe geometric registration for planar tiled displays is to obtain camera-projector homographies – either using a single camera [191, 266] or many [57]. In most cases, this homography-based approach is viewport-dependent, *i.e.*, the geometric correction is applied from the camera's perspective. In *Pixelflex* [266], keystoning is compensated with respect to four fiducial markers manually placed on the wall. Thus, the camera placement is more flexible.

If multiple projection images overlap, the recovered homography information can be used to calculate regions where multiple projections overlap. To compensate for the increased brightness in these overlap regions, intensity blending using linear ramps gradually attenuates pixels in overlap regions to create a seamless imagery [191].

If the display surface is non-planar, there are two common approaches to geometric compensation (*cf.*, [40, 48]): In the stationary view-dependent case, a single camera – ideally at the approximate viewer location – registers the imagery of a single projector (or multiple projectors) on an irregular surface, and a per-pixel mapping or triangle mesh (depending on the complexity of the surface) describes the individual pixels' target loca-

tions for an undistorted imagery (*e.g.*, [39, 49]). In the dynamic view-dependent case, a 3D reconstruction of the projection surface is required, as well as (head-)tracking of the viewer. In a two-pass rendering, the scene is first rendered from the user's perspective and then mapped onto the reconstruction of the projection surface using perspective texture mapping with the projector's projection matrix (*e.g.*, [189]).

However, if multiple users want to view and interact with the projection concurrently, such a view-dependent approach is not feasible. Raskar *et al.* [190] proposed the employment of the least squares conformal maps texture mapping technique [141] to achieve view-independent projection on arbitrary surfaces. This technique provides a distortion-free parameterization of developable surfaces with no angle-deformation and minimal stretch. For non-developable surfaces, such as a projection spanning two perpendicular walls and the floor, the algorithm aims at a minimal distortion. Displays thus appear like a wallpaper attached to the surface and are therefore particularly suitable for multi-user setups. Johnson and Fuchs [134] combine view-independent and view-dependent elements on a multi-planar cubicle setting.

Depending on the complexity of the surface, a fairly dense 3D mesh of the projection surface is required. However, common indoor geometries usually consist of piecewise planar surfaces, such as walls and tables. With a multi-planar representation of the surface, corners can be modeled precisely while dramatically decreasing the density of the polygon mesh. Quirk *et al.* [187] therefore fit planes into a fairly sparse point cloud reconstruction of indoor walls. Ashdown *et al.* [6] describe piecewise planar projection surfaces by plane-wise homographies.

Typical application areas for large projected displays are auditoriums or meeting rooms (*e.g.*, *Princeton Scalable Display Wall* [253], *PixelFlex* [266]). Irregular displays are often employed for edutainment and entertainment (*e.g.*, *CAVE* [66], *Smart Projectors* [39]), but also for personal workspaces (*e.g.*, *The Office of the Future* [192], *Escritoire* [8], or *Multi-Surface, Multi-Resolution Workspaces* [134]).

Large Display Content Creation

Projector systems, such as described in the previous section, require the desired output image to be corrected before being rendered. This may include geometric warping, blending of overlap regions, and photometric correction. Most examples mentioned in the previous sections use proprietary software for demonstrating their calibration results. They apply the required image transformations directly onto a rendered 3D scene (*e.g.*, [188]), use designated fragment shaders (*e.g.*, [40]), or employ a two-pass rendering approach (*e.g.*, [189]). However, if arbitrary output from any application or the final desktop composition needs to be corrected, the usual rendering pipeline has to be intercepted at some point to acquire and modify the anticipated image.

One approach to acquire the desktop content as bitmap images in a separate application is to facilitate tools like VNC [196] or Microsoft RDP. The primary purpose of these

applications is to send compressed desktop pixels from a server machine to a connected remote computer. Thus, they are commonly used for tiled display setups in combination with PC clusters, for instance in *PixelFlex* [266] or by Cotting *et al.* [64]. Whenever the display is modified and a desktop repaint is necessary, image data needs to be transfered from the graphics to the main memory, where warping and blending is applied by the render application. This introduces a considerable overhead if the tiled display is operated by a single machine, as for instance in the *display bubbles* infrastructure [65].

DeskAlign [255] uses a dual graphics pipeline infrastructure to render the unmodified desktop on the first pipeline and then passes the pixels to the texture memory of the second pipeline, where the image is warped. Thus, for running a two-projector display, a quad graphics card is required.

Large Display Interaction Techniques

With the raise of large-scale displays, new user interaction challenges have to be addressed. Robertson *et al.* [200], as well as Czerwinski *et al.* [69], list many of these challenges and present new user interface prototypes to overcome these problems. Roughly summarized, the identified problems evolve around window and task management (*e.g.*, placement of popping up windows, managing a large number of windows), information perception (*e.g.*, tracking the cursor, perceiving distant information), interaction at a distance (*e.g.*, accessing elements out of the arm's reach), and the discontinuity emerging from combining multiple monitors to a common interaction space. The issue of discontinuity will be discussed in more detail in the next section on multi-display environments (Section 2.2.3). Task management techniques have been presented in combination with overlapping window management in Section 2.1.2. Window management techniques, which have been designed in particular for large-scale displays (*e.g.*, overlap avoidance [27] or occlusion-aware interfaces [251]), were discussed in Section 2.1.2.

Research on perception showed that only a very small portion of the display is in the active user's visual field [260]. Thus, the user can easily miss important information on the screen. A prominent example is that users cannot keep track of the cursor on a large display [68, 200]. To better locate the cursor, motion trails have been added if the cursor is moved very quickly [19], a *spotlight* metaphor was introduced, which dims the entire display except for the area around the cursor [138], and cursors were visually *anchored* to their "home" device when being redirected to an adjacent display [195]. Hoffmann *et al.* [108] designed visual cues to guide the attention to the currently active window after performing a window switching operation.

To facilitate the access and operation of information artifacts located at a distant display location, several interaction techniques have been developed. They basically aim to satisfy two requirements: being able to access items beyond arm's reach when using direct input devices and overcoming *Fitts' Law* [78, 144]. Fitts' Law predicts movement times as function of the distance to a target and the target's size. The *index of difficulty*

(ID) describes targeting difficulty as logarithmic relation of the required travel distance and the target width [144]:

$$ID = log_2 \frac{2A}{W}, \tag{2.1}$$

where A is the movement distance (or *amplitude*, as inspired by Shannon and Weaver's theorem [216]) and W is the target width. For large displays, this index of difficulty implies that acquiring targets generally requires an increased movement time due to the increased number of pixels. Balakrishnan [12] investigated different interaction techniques aiming to "beat" Fitts' Law, by either reducing the distance to the target, expanding the (virtual) width of the target, or applying a combination of both (*i.e.*, dynamically changing the control-display gain). Large display reaching techniques usually reduce the distance to the target. Examples are *Frisbee* [137], a portal technique for interactive whiteboards, *drag-and-pop* and *drag-and-pick* [17], which bring suitable proxies of distant icons close to the cursor dragging an item, and *Vacuum* [30], which improved the drag-and-pop/pick concept. The *dollhouse metaphor* [234] does not reduce the distance to a particular target or screen region, but rather scales down the entire display to a small-scale navigation space, where the user can control the large display. However, as the target size is reduced together with the target distance, the index of difficulty remains effectively unchanged.

2.2.3 Multi-Display Environments

Multi-display environments (MDEs) combine multiple displays of various form factors into a single interaction space. They may comprise conventional monitors, portable devices, large wall displays, as well as tabletop projections. Examples of MDEs can be found in control rooms (*e.g.*, [241, 254]), meeting rooms (*e.g.*, [129, 225]), or office environments (*e.g.*, [37, 192]). These application areas illustrate that MDEs are usually employed in collaborative settings, often combining personal displays and large-scale public displays. Only a few systems have been developed with single-user workspaces in mind (*e.g.*, *Kimura* [143]).

Mind that we consider discontinuity between the displays and heterogeneity of the individual displays as criteria for an MDE, not only the combination of multiple display devices into a common viewing and interaction space. Otherwise, conventional multi-monitor settings, tiled displays (*cf.*, Section 2.2.2), and focus+context screens (*cf.*, Section 2.1.1) could be considered as MDEs as well.

Multi-Display Control and Input

By reviewing the literature on MDEs, four different viewing and interaction space approaches using multiple displays can be discovered:

Emulating a seamless interaction and viewing space. Systems of this category aim to minimize the individual display factors by adjusting the viewing space towards the user's point of view. They aim to create the illusion of a seamless display

spanning across multiple surfaces, as, for instance, in a *CAVE* [66]. As an example, *E-Conic* [169] perspectively corrects individual application windows to align with the user's field of view. Independent of the display the window is currently associated with, the window always appears to be floating in front of the user. As windows are arranged on a virtual plane, they can also span multiple output devices, where each device shows an oblique portion of the window. Conceptually, this category aims to fulfill the same requirements as dynamic view-dependent projected displays (*cf.*, Section 2.2.2) – but additionally extends this concept to multiple discontinuous displays.

Separate interaction and viewing spaces. These systems explicitly exploit individual display factors for showing appropriate information on distinct displays, as well as providing suitable input techniques. For instance, in the *iLAND* system [230], users can interact with a touch-sensitive table, a digital whiteboard, and tablet PCs embedded into chairs. Tangible objects facilitate information exchange between these displays.

Table-centric interaction spaces. Like in the previous example, visual information is adapted to the individual display factors. However, input cannot be directly issued at each display device and is rather provided from a central point. A tabletop display serves as a hub to access information on the peripheral wall displays. Instead of physically moving to the respective display, the user can remain seated at the tabletop while controlling a remote display. For instance, Wigdor *et al.* [265] use a central touch-sensitive table to control the entire environment. To access and transfer information on other displays, miniature views of adjacent displays are shown at the tabletop boundaries.

Personal device-centric interaction spaces. Similar to table-centric interaction spaces, the user interacts with the entire environment using a single device – her indirect pointing device (such as the mouse) attached to her personal workspace (such as a laptop or workstation PC). In contrast to table-centric interaction spaces, the MDE infrastructure enables seamless interaction across all display devices without physical movement of the user by facilitating indirect pointing devices. In contrast to the first category, the viewing spaces are independent and separated. For instance, in *Augmented Surfaces* [195], the users control a cursor "anchored" to the trackpad of their notebooks. Moving the cursor beyond the notebook's monitor edge extends it to the underlying table and further to an adjacent wall display. Objects can be relocated by "hyper-dragging" them across the display boundaries.

Table 2.1 shows a comparison of selected MDE systems with respect to the above listed categorization. Example infrastructures are listed in chronological order, together with their primary target application area and their input technique(s).

	Examples	Application Area(s)	Input Technique(s)
Seamless spaces	E-Conic [169]	not specified	mouse [170]
Separated spaces	CoLab [225]	meeting room	mouse (monitors), pen (wall)
	iLAND [230]	meeting room	touch, pen (table, wall, tablets)
	Kimura [143]	single-user workspace	mouse (monitor), pen (wall)
	Interactive Workspaces [128]	meeting room	pen, touch (wall), mouse [129]
	MultiSpace [74]	meeting room	touch (table, wall), mouse
Table-centric	Table-centric spaces [265]	meeting or control room	touch (table)
	WeSpace [264]	meeting room	touch (table)
Personal device-centric	Courtyard [241]	control room	mouse
	Augmented Surfaces [195]	office	mouse
	multi-cursor X [254]	control room	mouse
	Swordfish [96]	not specified	mouse
	IMPROMPTU [37]	office	mouse
	Deskotheque [181]	office	mouse

Table 2.1: Comparison of selected MDE systems.

As the MDE presented in this thesis (*Deskotheque* [181]) aims to extend personal workspaces to a shared interaction space in an office environment, we do not discuss the first three categories in more detail here. Also, as most other example systems in the category of personal device-centric MDEs, we focused on mouse interaction as the primary input technique. Therefore, alternative input techniques, such as touch-sensitive surfaces, pen-based input, gesture-based input, speech-based input, or tangible user interfaces, will not be discussed in this thesis. An overview on input techniques can be found in the survey by Hinckley [106].

In our MDE, we treat the viewing space on multiple displays as discontinuous information space. As in all of the MDE systems of the last three categories in Table 2.1, we employ dedicated information sharing mechanisms to relocate information artifacts (*e.g.*, application windows) across displays. According to the taxonomy of multi-surface interaction and visualization techniques by Shen *et al.* [217], this mode is called "independent": Content and user input is not coordinated across the surfaces, but content relocation across the surfaces is possible. The other two modes presented in this taxonomy are "reflective" (all surfaces show the same information) and "coordinated multi-view" (all surfaces share the same data but with a different representation). The latter mode is especially interesting for multiple coordinated views (*cf.*, Section 2.1.1) operated in a multi-display environment. As an example, Forlines and Lilien [80] employed multiple coordinated views of a protein on a touch-sensitive table, two wall displays, and a tablet PC for fine-grained interaction. As in a conventional multiple coordinated view system, changes in the point of view on one view are propagated to the other views and displays, respectively. The appearance of the views on the individual displays can be set independently.

Mouse Pointer Navigation in Multi-Display Environments

The most common approach to create a seamless interaction space in a personal device-centric environment is to facilitate indirect pointing devices like the mouse. Mouse pointer

navigation techniques for MDEs thereby do not only need to assure efficient on-display navigation, but also allow for redirection of input control across display boundaries. A popular approach to achieve seamless cross-display mouse pointer navigation is derived from conventional multi-monitor systems. Adjacent display edges are virtually connected (or "*stitched*" [170]) to create static mouse pointer paths (*e.g.*, *Anchored Cursor* [195], *PointRight* [129], *Swordfish*'s bindings [96], or *Synergy* [212]). Whenever the mouse pointer reaches a display edge, which is virtually connected to a remote display, input is redirected to the associated target display.

MouseEther [18] additionally incorporates visual discontinuities introduced by monitor bezels and display size-resolution mismatches into the device space. An experiment [18] has shown that MouseEther decreased movement times if the size-resolution mismatch is high. *Perspective Cursor* [170] extended this approach by evaluating mouse input events from a tracked user's perspective of the environment. Thereby, it additionally introduces a non-uniform control/display (C/D) gain when navigating within a single display, caused by perspective foreshortening, if the user is not sitting directly in front of the display. The authors showed that a perspective control space could improve targeting tasks across complex display arrangements, where an intuitive edge-to-edge mapping was not possible. However, Nacenta *et al.* [168] later showed that warping the mouse across a physical gap (*i.e.*, redirecting the mouse directly to a target display when reaching a virtually connected edge without virtual movement in display-less space) between two monitors results in a higher performance compared to a MouseEther, if the physical distance between the monitors is large. In addition, they showed that steering tasks in heterogeneous MDEs using the Perspective Cursor require a perspectively corrected viewing space – otherwise conventional stitching without C/D gain adjustments of the control space is more appropriate [169].

An alternative to seamless mouse pointer navigation spaces is to invoke transitions through an explicit trigger, for instance by pressing a dedicated button on mouse or keyboard (*e.g.*, as in M^3 [28, 29]), by selecting the target display in a GUI list (*e.g.*, *Mighty Mouse* [44]), or a miniature view of the environment (*e.g.*, *ARIS* [33]). These techniques are often referred to as *pointer warping* [29]. Investigations have shown that pointer warping is beneficial when crossing multiple homogeneous monitors [28], accessing displays with strong size-resolution mismatches [29], and when sitting at an inconvenient location towards the display [257].

Another alternative to a seamless interaction space is to provide content redirection (*i.e.*, forwarding the visual content of a display) instead of input redirection (*i.e.*, forwarding the input control to a remote display). An early example was *Semantic Snarfing* [164], where users could indicate a region of interest on a remote screen by using a laser pointer. This screen region was then transfered to their handheld personal device. Some MDE systems use content redirection as alternative control facility, in addition to input redirection. In *WeSpace* [264], users can forward their laptop content to a shared display wall, and can control the layout of multiple laptop views on this display (*e.g.*, side-by-side comparison

or overlaying [127]). Others use portals or world-in-miniature views to relocate content or control the layout of information artifacts on remote displays. For example, the shared display of *IMPROMPTU* [37] can be controlled by a world-in-miniature view, as well as by redirecting the input. Wallace *et al.* [257] showed that content redirection is more efficient than input redirection if the seating arrangement is non-optimal and the user has to turn head and body to see the remote display. On the other hand, Hawkey *et al.* [101] showed that content redirection is less suitable for co-located collaboration. Using a tablet PC to provide direct input on shared display content lead to a loss of a shared visual reference and, as a consequence, a loss of shared understanding.

Information Transfer across Displays

A similarly important task than forwarding the input to a remote display for remote control of objects is to transfer information items (*e.g.*, application windows, pictures, or other multimedia elements) across discontinuous displays. Nacenta *et al.* [166] grouped the sub-tasks for cross-display information movement into four categories: *putting* (moving the object to a remote display), *placing* (positioning the object at a precise location on the remote display), *manipulation* (moving the object within the remote display), and *getting* (fetching an object from a remote display).

There are several techniques that rely upon direct pointing devices or auxiliary devices for information transfer. Examples include *Pick and Drop* [193], which facilitates a pen to pick up an item and drop it at a remote location; the *passage* mechanism [230], where users can assign information to any physical object placed on a *bridge* located at each display and release it at a remote bridge; or *Touch Projector* [45], where objects on a public display can be selected on a mobile phone touch screen and dropped on a remote display by pointing the phone towards the target display surface. However, as the primary input mechanism of the Deskotheque project presented in this thesis relies on the standard mouse device, the focus of this section will be on information transfer facilitating indirect pointing devices, like the mouse.

Given a seamless mouse pointer navigation frame across displays, objects can be relocated by simply dragging them across the display boundary to a remote display, following the mouse pointer paths. A first example of this *hyperdragging* technique was presented by Rekimoto *et al.* [195] for the *Augmented Surfaces* environment. Similar to hyperdragging, users can drag application windows across display boundaries in *E-Conic* [169]. As E-Conic aims to compensate for any discontinuities across displays, the user performs simple drag-and-drop operations on her "perspective image plane", irrespective whether the *Perspective Cursor* [170] is currently within a visible display region or in display-less space and therefore invisible.

Hyperdragging and similar techniques can lead to considerable physical and mental effort for the user – especially if the travel distance is large (in the device space), the display setup is rather complex, so no intuitive mouse pointer paths can be created, or multiple

objects need to be relocated. To overcome large distances, drag-and-drop techniques for large displays (Section 2.2.2) can similarly be employed for drag-and-drop across discontinuous displays. In addition, GUI-based techniques have been developed, which do not necessitate the user to overcome the entire travel distance with the pointer. There are two common GUI-based relocation techniques: lists, where individual displays are represented as textual items or icons, and world-in-miniatures [227], where the entire environment is represented in a scaled-down view, allowing the user to perform simple cross-display tasks. In the *iRoom* [128], users could relocate websites by selecting a target display from a context menu list embedded in the browser [130]. In *ARIS* [33], users can relocate windows by invoking a miniature iconic view of the environment from the desired window's title bar. Windows can then be dragged and dropped across displays within the miniature view, which is reflected on the windows in the real environment. *SEAPort* [35] extended this concept by putting more emphasis on information discovery: instead of miniature window textures, application icons were employed within the miniature view and users could interactively request a non-occluded view of all windows on the selected display. In the table-centric MDE of Wigdor *et al.* [265], lateral wall displays can only be operated from the central tabletop display. To relocate objects to the wall displays, the user drags objects from the center of the table to miniature views of the wall displays, located at the closest boundaries of the table. *IMPROMPTU* [37] provides multiple interfaces to share application windows. Docks of available collaborators and a single shared display provide a list of shared remote application windows, which can be duplicated to the user's local machine. In addition, a miniature view of the shared display allows the users to re-position remote windows.

A comparison of multi-display display reaching techniques for pen-based devices has shown that radar views (a world-in-miniature technique) had superior performance for cross-display object relocation [167]. *Pick and drop* [193] was found to be acceptable, if the targets were located within hand's reach [167]. For mouse-based interfaces, Biehl and Bailey [34] compared a hyperdragging technique with ARIS [33] and a text-based interface for an application relocation task. They showed that ARIS was more effective than the text-based interface and had equal performance with hyperdragging. However, for simple input redirection, seamless mouse pointer navigation was superior to ARIS.

In contrast to hyperdragging, GUI-based relocation techniques decouple the information transfer task from input redirection. While advantageous in some situations, fine-grained placing and manipulation on the remote display requires an additional input redirection activity. In addition, users need to mentally map the textural or pictorial representation of the environment onto the real world [166].

2.2.4 Middleware

Displaying information across multiple output devices often requires a PC cluster. Especially MDE systems for meeting rooms or office environments have to rely on a distributed

system architecture to accommodate for the users' private workstations, as well as shared interaction spaces. Typically, two major tasks have to be accomplished by a middleware for distributed display environments: information sharing and input redirection (often in combination with multi-pointer support). In addition, the creation and management of a spatial display model will be discussed, as this is a main distinguishing aspect of the Deskotheque environment presented in this thesis.

There are numerous additional tasks a middleware for distributed display environments may deal with, such as data and security management (*cf.*, issues in ubiquitous computing [262]). However, the focus of this thesis lies on visual information management. Therefore, these aspects will be neglected throughout this thesis.

Information Sharing

Sharing information across multiple machines is often facilitated by specialized ubiquitous computing middleware, such as *Gaia* [205], *Roomware* [186], or *iRos* [184]. Although some of these environments aim to simplify legacy application support by keeping the amount of required modifications to a minimum, unmodified applications cannot be operated in such a specialized environment.

Other systems rely on VNC [196] for sharing individual application windows or entire screens. The middleware [207] used for *E-Conic* [169] renders application windows side-by-side on an application server machine. On each display client, a VNC client receives the remote desktop image, the respective application windows are cropped from the image, and rendered as transformed textured quads in a fullscreen OpenGL application. In *WeSpace* [264], the entire laptop screens of the collaborators are forwarded to a shared display using VNC. By using VNC, legacy applications can be employed in a distributed MDE without modifications. However, when using a central application server, the available software is restricted to installed applications on the server machine. Users cannot facilitate specialized software on brought-in personal devices. When using VNC to share the entire (private) screens, users are not restricted to pre-installed software, but cannot control which application windows are shared among the collaborators, which may interfere with privacy concerns.

Wallace *et al.* [254] used a more flexible approach to peer-to-peer window sharing: They employed *XMove* [221] to dynamically change an X window's server by exploiting the inherent network transparency of the X Window System. The X Window System allows any X application (X client) to be operated by an X server on a remote machine by providing a fully network transparent interface. However, X does not allow applications to change the X server at runtime. XMove operates as a so-called pseudoserver and thereby enables an application window to be migrated to a remote host at application runtime. The actual migration is transparent to the X server, as well as to the application itself. The same approach for window migration was employed by the first prototype version of our *Deskotheque* environment [181]. However, as XMove is limited to the X Window

System and has not been maintained for a long period of time, Wallace *et al.* [256] later presented *SharedAppVNC*, which shares single application windows based on the VNC protocol across platforms. For the X Window System, *x11vnc*[‡] allows for sharing of a particular X window instead of the entire desktop.

An alternative to sharing individual windows is to create a virtually large desktop spanning multiple displays and machines, respectively, by facilitating multi-head support for multiple machines. For the X Window System, *Distributed Multihead X*[§] (Xdmx) distributes a single X session to multiple heads and machines, respectively, by employing a proxy X server. Unfortunately, Xdmx turned out to be incompatible with the X Window System's *Composite* extension, which is required for compositing window management – a core concept utilized for this thesis (*cf.*, Section 2.3.1). *Chromium* [111] can be used to operate existing OpenGL applications in a distributed system. The latter approach is commonly employed for building seamlessly tiled displays (*e.g.*, [220, 224]).

Recent advances of the X Window System, which facilitate the Composite extension, do not fully overcome the problems of distributed interactive spaces, but indicate a promising new direction. For instance, *xpra*[¶] allows sharing of individual X windows and changing window assignment at runtime. However, the work is still in progress and not stable yet. *Wayland*[‖] accomplishes compositing of individual application windows or entire X screens in a more efficient way than the X Window System in combination with a compositing window manager. As Wayland is meant to be light-weight, remote rendering is not (yet) supported.

Input Redirection

Input redirection is concerned with the forwarding of input events to a remote machine. Examples of commercial and freely available input redirection tools are *x2x*[**] for the X Window System, *Desktop Rover* [171] for Microsoft Windows, and *Synergy* [212] for cross-platform input redirection. In the pioneer MDE system *Courtyard* [241], a centralized *user interface management system* grabs input events from personal devices, translates the mouse pointer position to the shared display's coordinate system, and renders the pointer on the shared display. *PointRight* [128] facilitates the event heap of *iRos* [184] to forward pointer and keyboard events from a sender to a receiver in a peer-to-peer fashion. *Mighty Mouse* [44] uses the VNC protocol and *x2vnc*[††], respectively, for input redirection.

[‡] `http://www.karlrunge.com/x11vnc/`
[§] `http://dmx.sourceforge.net/`
[¶] `http://code.google.com/p/partiwm/wiki/xpra`
[‖] `http://wayland.freedesktop.org/`
[**] `https://github.com/dottedmag/x2x`
[††] `http://fredrik.hubbe.net/x2vnc.html`

Multi-Pointer Support

If multiple users want to interact on a single display and machine, respectively, a middleware is required to coordinate concurrent input from multiple input devices. Mostly, multi-user interaction is coordinated within specialized groupware applications (see Section 2.1.3 for examples), or "retro-fitted" and wrapped applications [79, 119].

When using single-user legacy applications (*i.e.*, applications which cannot handle or discriminate input events from multiple input devices) and standard desktop environments, "floor control policies" assign control to multiple input devices attempting to interact concurrently. Some input redirection tools presented in the previous section are simply limited to a single input device for controlling the entire environment (*e.g.*, *Synergy* [212]). *PointRight* [129] and *Mighty Mouse* [44] support multiple input devices in the environment but only allow one input device to operate one display and machine, respectively, at a time. In *Courtyard* [241], input events on the shared display are sequentially handled, irrespective of the pointer issuing the event.

The *multi-cursor window manager* [254] for the X Window System handles multiple input devices on the window management layer. The window manager captures input events from remote devices using x2x, generates X input events for the local X Window System, and encodes an ID for the remote pointer into an unused status field of the X event. Remote cursors are rendered as color-coded cursor shapes by the window manager, which also handles focus window assignments. The multi-cursor window manager is running on a shared Linux PC, while remote pointers can be contributed from different platforms using x2x. Disadvantages of the approach are the limitation to eight pointers and the restriction to the specialized window manager.

Multi-pointer X (MPX) [115] handles multiple local input devices on windowing system level and is therefore not restricted to a specialized window manager. The original implementation of the X Window System is principally able to handle multiple input devices, but merges all input events into a single event stream and pointer, respectively. MPX assigns each input device a dedicated pointer on the screen and generates uniquely identifiable *XInput2* events, *i.e.*, input events using a modified version of the X *Input* extension. For compatibility with single-pointer legacy applications, each input device additionally generates *core* input events, so single-pointer applications can receive the respective input events but will not discriminate their origins. The *multi-pointer window manager* [115] was developed to demonstrate personalized annotations and floor control capabilities of MPX, but is not required for MPX operation. The *device shuffler* [115] enables a simple version of input redirection to support multi-machine configurations. For more flexible input redirection support and cross-platform configurations, Synergy [212] was extended to support concurrent interaction on shared hosts by facilitating MPX on Linux machines [71] (see Section 2.3.2 for more information).

Spatial Model Configuration

Both, information sharing and input redirection require a spatial description of the displays for proper operation in a spatially consistent manner. In addition, cross-display visualizations, such as presented by Mackinlay and Heer [146] or *OneSpace* [200], need information about the spatial display arrangement.

In the simple case of a co-planar multi-display setting, a 2D spatial description of the environment is sufficient. For instance, to calibrate a dual-monitor setting for *MouseEther* [18] or *OneSpace* [200], the user needs to manually align an image with an arrow across the monitor bezels. *Stitching* [107] allows the user to establish a network connection across two tablet PCs by drawing a stroke starting from one device to another using a pen. The direction of the outgoing and incoming stroke infers the spatial relationship of the two tablet devices, which can be facilitated to stretch images across adjacent devices or for object relocation tasks.

More complex MDE setups cannot rely on such a simple 2D description. *PointRight* [129] and *Synergy* [212] employ user-defined configuration files containing a textual description of exit and entry display edge intervals for input redirection. This way, pair-wise 2D relations between adjacent displays are established. Alternatively, a single 2D mapping of all displays can be produced (as, for instance, in the *stitching* condition of [170]). However, in this case, complex display arrangements may lead to unintuitive transitions. In *Swordfish* [96], users can establish these display-connecting edge intervals individually at runtime. Thus, the system itself does not need to automatically create a spatial model of the environment. As the perspective mouse pointer navigation frame of *Perspective Cursor* [170] relies on the user's location within the environment, a detailed 3D model of the environment, as well as constant user tracking, is required. For this technique, the location of the stationary displays were manually measured offline, while the user's head pose and location of mobile displays were determined with a 3-DOF tracking system [165]. To create the 2D fold-out representation of the environment of *ARIS* [33], the user is provided with a GUI tool to manually construct the world-in-miniature view [36].

For the automatic creation of the spatial model in the Deskotheque environment, we rely on techniques that are mainly employed for geometric compensation of irregular and tiled projected displays (*cf.*, Section 2.2.2). Using a camera-assisted calibration technique, we cannot only determine the displays' positions within the environment, but also a detailed geometric description of their projection surfaces. Thus, we are enabled to flexibly combine monitors and irregular or tiled projected displays into a common interaction space, with fast re-calibration facilities.

2.2.5 Discussion

In the following, a short discussion will summarize the most important aspects of related work in display environments with respect to this thesis. It will start by reviewing the (software) system design of MDEs, the importance of spatial awareness, and finally the

design of window and information management in such environments.

System Design of Multi-Display Environments

In terms of system design, MDE infrastructures usually fall into two categories: They either aim to support legacy applications (or minimally modified legacy applications) or they put their focus on presenting entirely novel applications, tailored towards collaborative interaction and visualization on multiple surfaces. Examples for the second category are *coordinated multi-view* applications [217], which show multiple coordinated views of a shared data set distributed to multiple displays (*e.g.*, [79, 80]).

Satisfying the requirement to support legacy applications is important, if the primary goal of the infrastructure is to engage a large audience. Users usually prefer their powerful and well-known applications to specific collaboration-aware or multi-display aware counterparts [90, 140]. In addition, information analysis and management often requires a plethora of information sources and supportive applications. A dedicated software infrastructure, tailored to a specific environment and task, cannot easily fulfill these requirements.

However, today's windowing systems and window managers were not designed for emerging display environments and collaborative interaction. Important *mechanics of collaboration* [95] and visual coordination techniques to support information management on large displays, irregular displays, or multiple displays, need to be considered in current systems.

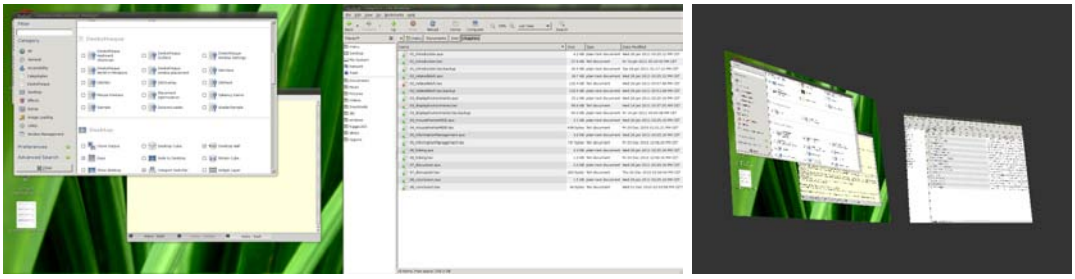
The MDE design presented in this thesis (Chapter 3) takes these considerations into account. The system design assures that legacy applications can be operated without any modifications. Basic functions of an MDE, such as geometric compensation of projected displays, seamless mouse pointer navigation, multi-pointer coordination, information sharing, and fundamental visual coordination, are accomplished transparently to legacy applications. In addition, a light-weight API allows MDE-aware applications to query user identities and to coordinate user selections across multiple views of the same application or multiple applications.

Spatial Awareness in MDEs

As the user's viewing and interaction space is expanded by the availability of multiple large displays, spatial cognition is increasingly important. Ha *et al.* [97] discovered that the spatial relationship towards the displays is the main consideration of users when organizing their workspace. There is evidence that incorporating the spatial display arrangement into the cross-display mouse pointer navigation frame increases performance [18, 34, 170].

Spatiality is not only an important aspect in terms of navigation. Scott *et al.* [215] demonstrated how users facilitated space on tabletop displays to arrange information artifacts. Others showed that users explicitly facilitate focus and peripheral display regions for primary and secondary information resources [32] and use display discontinuities for

task separation [91]. However, spatial awareness plays a minor role in the design of information presentation in MDEs. Usually, the user has to provide the spatial information manually to enable certain capabilities, such as spatially consistent navigation. Apart from the *E-Conic* prototype [169], window managers are usually unaware of spatial display configurations. As an example, see Figure 2.2: the window manager usually considers the boundaries between adjacent displays, but neither reflects their physical size, their distance, nor their orientation.



(a) (b)

Figure 2.2: Spatial awareness in window managers: (a) Usually, the window manager treats multiple output devices as one seamless rectangular screen. (b) The world-in-miniature view (using [24]) shows that the right monitor is actually slightly smaller (although it has the same number of pixels) and that the spatial arrangement is not planar.

The Deskotheque MDE infrastructure presented in this thesis automatically creates a fine-grained 3D model of the environment as foundation for interaction and presentation techniques (Chapter 4). The spatial information recovered from the calibration step is used for providing undistorted (tiled) projected displays (Section 7.3.2), display-adaptive window placement and size (Section 8.3), spatially consistent multi-display mouse pointer navigation (Chapter 5), as well as visual links connecting related items across multiple displays (Section 8.1).

Window Management for Large Displays and MDEs

As multi-monitor arrangements are becoming more common, there have already been attempts towards more suitable multi-monitor window management. For instance, maximizing to the extent of a single output device (see also Figure 2.2(a)) or snapping a dragged window at output device boundaries are now features of many common window managers. However, these features are not feasible on very large, seamless displays, where no pre-defined partitioning due to (visible) output device boundaries is given. Still, researchers have shown that users manually apply a segmentation to their visual workspace, such as keeping important portions of windows visible while avoiding strict tiling [113], creating focus and context regions [32] or semantic clusters of windows [4]. Exploratory studies have indicated that users prefer large displays to their smaller counterparts, even though they invest considerably more time and effort in window management [4, 32].

In addition to the lack of support for large, unpartitioned visual space, window managers do not consider emerging display factors caused by the employment of projectors. Examples are non-vertical displays, non-rectangular displays, and non-planar displays (Section 1.1).

Therefore, one idea that will be explored in this thesis is *display-adaptive window management* (Section 8.3). In this concept, the window manager is aware of the physical display properties and dynamically adjusts window locations and size to the prevalent display factors. The aim is to minimize manual window management overhead caused by a large number of pixels, irregular display outlines, and physical discontinuities.

2.3 Enabling Technologies

This section will shortly introduce enabling technologies, which were facilitated for the prototypes created for this thesis.

2.3.1 Compositing Window Managers

3D accelerated compositing window managers are available on most common platforms, such as *Quartz Extreme* on Mac OS X, Microsoft Windows Vista's *Aero*, and OpenGL-based implementations for Linux OS. They provide compositing effects like window blending or rendering of the desktop onto an interactive 3D cube. In contrast to these merely eye-catching window management techniques, *Metisse* [55] was developed with HCI researchers in mind. Metisse facilitates three major X Window System extensions for OpenGL window compositing: *Composite* (off-screen rendering of windows), *Damage* (window region update notification), and *Event Interception* (input event interception and transformation). Windows are rendered within an OpenGL scene and can be arbitrarily transformed. OpenGL picking and the event interception extension are employed to transform registered cursor locations to the original window locations, which are requested by the X Window System for event processing. A light-weight scripting language allows for easy creation of window management functions, such as automatically rotated windows or peeled back windows.

The well-known compositing window manager Compiz for Linux provides a plug-in architecture to easily apply new windowing effects without changing the core implementation. All window manager extensions presented in this thesis were developed for Compiz (version 0.8.4) or its predecessor Beryl. The reasons to prefer Compiz to Metisse were mainly based on technological considerations: Compiz is now the default window manager for the major Linux distribution Ubuntu and therefore heavily maintained. In contrast, some features of Metisse are no longer supported on current versions of the X Window System. For instance, the event interception extension used in Metisse has not been maintained for a long time and was finally removed from the X Window System when MPX

was introduced [‡‡]. In addition, the plug-in infrastructure of Compiz provides a richer development platform for highly sophisticated window manager operations, compared to Metisse's light-weight scripting approach.

Similar to Metisse, Compiz uses the *Composite* extension of the X window system to render application windows into an off-screen buffer on the graphics card. The window manager can access these pixmaps as textures using the OpenGL extension `EXT_texture_from_pixmap` – either directly via the NVIDIA driver or indirectly via the Xgl implementation. Thus, 3D compositing window managers make effective use of available graphics hardware and provide efficient ways to manipulate desktop and window geometries, as well as their textures.

In addition to conventional window manager tasks (such as moving or resizing windows), more advanced operations on the window layer (*cf.*, Figure 1.4), such as geometric transformations or transparency effects, can be easily achieved by accessing and modifying the window's quad or texture, respectively. As an example, a novel window management approach for increasing the visibility of important information on the screen is presented in this thesis (Section 7.4). For operations on the screen layer, the Compiz window manager enables us to access the final desktop composition in a plug-in and to render the desktop image into a frame buffer object. The buffer content is then texture-mapped onto custom geometry, which can be arbitrarily modified. As an example, we present warping and blending for projector-based displays as window manager extension (Section 7.3.2). Furthermore, Compiz allows us to render arbitrary information on top of the desktop. We facilitate this for a cue-based focus and context technique to visually filter information on a large display (Section 8.1).

2.3.2 Input Redirection and Multi-Pointer Support

For input redirection, we relied on the cross-platform mouse pointer sharing tool *Synergy* [212]. In its original implementation, Synergy enables a single mouse and keyboard pair to be shared across multiple machines. Display connections are described as connected edge intervals, specified either in a configuration file or using a GUI tool. For spatially consistent mouse pointer navigation in the Deskotheque environment, two major modifications were required:

1. A navigation framework on top of Synergy applies input redirection triggers and mouse pointer movement corrections based on the automatically created model of the environment (Chapter 5).

2. *Synergy+MPX* [71] extends Synergy to be multi-pointer aware, using MPX [115] (*cf.*, Section 2.2.4). It allows for flexible client-server configurations so multiple pointers can be dynamically redirected to a single machine and interact on the host concurrently. The current implementation is limited to Linux servers and clients.

[‡‡] `http://freedesktop.org/wiki/Software/XEvIE`

However, with minor protocol adjustments, single-pointer servers of different platforms can contribute to multi-pointer Linux clients as well. Figure 2.3 illustrates the differences between the original Synergy implementation and Synergy+MPX.

(a) (b)

Figure 2.3: (a) A configuration of the original Synergy implementation: sharing a single mouse and keyboard pair across multiple machines and platforms (`http://code.google.com/p/synergy-plus/`). (b) A possible configuration of Synergy+MPX with multiple mouse and keyboard pairs shared across multiple Linux machines.

Part II

Multi-Display Framework

Chapter 3

System Infrastructure of the Deskotheque Environment

In this chapter, the multi-display framework Deskotheque will be presented as a foundation for further interaction and information presentation research on MDEs and irregular displays. The aim of the Deskotheque environment is to extend personal workspaces in conventional office environments to a shared interaction space. Unused wall and table spaces in offices are turned into interactive surfaces by employing multiple projectors.

3.1 Requirements and Design Principles

To allow for a flexible usage in existing office spaces, a number of requirements were formulated:

Collaboration transparency. The environment should allow multiple users to interact concurrently on shared information spaces. At the same time, privacy of personal workspaces has to be assured.

Application transparency. To support users in their everyday activities, the environment should not restrict the user to specific applications, tailored to multiple displays or multiple users. Instead, the environment has to allow the operation of legacy application windows on personal workspaces, as well as on shared information spaces. Furthermore, the environment has to ensure that applications and control can be transfered between the displays.

Distributed system. The environment should be able to incorporate all personal workspaces available in the environment, as well as all shared projected displays, into a common interaction space. Therefore, a distributed system infrastructure is required. Supporting multiple platforms is desirable, but was not seen as a major goal and was therefore not considered in the system design.

Commodity hardware. To allow for a wide-spread adoption, operation of the environment should not require special-purpose or extraordinarily expensive hardware infrastructure. Therefore, we refrained from employing rear-projection screens, touch-sensitive surfaces, or tracking systems.

Spatial adaptivity. The environment should be able to deal with manifold spatial display arrangements, in particular irregular projection surfaces caused by physical unevenness or unconventional aspect ratios. It has to be possible to flexibly arrange multiple projectors, which can be discontinuous or overlapping, to create shared interaction spaces, even in office spaces with limited space resources.

Based on this set of requirements, we derived the system design for the Deskotheque multi-display framework. Deskotheque was designed as personal device-centric interaction space (see Section 2.2.3). The user is assumed to control the entire environment from her personal device, located in an office room (see Figure 1.1 as an example). Usually, this personal device is a PC connected to one or multiple monitors, or a laptop. We assume the user operates the personal device by mouse or touchpad, and a keyboard. Although personal handheld devices with touch-sensitive input, like tablet PCs or smart phones, are becoming more ubiquitous, we considered office workspaces to be primarily stationary and limited input support to indirect pointing devices. Figure 3.1 shows an early sketch of a meeting room MDE.

Figure 3.1: An early sketch of the anticipated MDE in a meeting room.

The core feature of the Deskotheque environment is a fine-grained description of its spatial arrangements – *i.e.*, the geometric and topological properties of the display environment. The model is created in an offline camera-assisted calibration step, which is described in more detail in Chapter 4. At runtime, the spatial information is utilized for spatially consistent cross-display navigation (Part III) and environment-aware window management techniques (Part IV).

The system infrastructure and user interface prototypes of the Deskotheque system presented in this thesis aim to support users in *visual* information management. Of course, a fully functional MDE additionally requires distributed data management and security management. These aspects will be disregarded in this thesis. For instance, when referring to information sharing, we only address sharing of the visual presentation, while the underlying data remains on its original host. Also, we do not provide any mechanisms to prevent users from modifying or deleting data associated with another user's shared visual presentation.

3.2 Distributed Software Infrastructure

Deskotheque is implemented on top of the X Window System for Linux. It is tightly integrated into the windowing system and therefore allows for application- and network-transparent operation.

Deskotheque employs a distributed system involving multiple computer nodes (*"modules"*) in a network. Modules may represent a single-user workstation consisting of a monitor with mouse and keyboard input, a tiled projected display without any input capabilities, or a self-contained device, such as a laptop. Individual modules are coordinated by a central master instance, which is responsible for configuring the individual modules and to trigger certain components at runtime. It is also responsible to create and manage the spatial model.

The framework is sub-divided into several components, which operate largely on windowing system level. When designing these components, we obtained application transparency by leveraging existing components of the Linux and X Window System software infrastructure. Each component was extended to serve as client application to the Deskotheque runtime communication, which uses the distributed object library *Ice*[*]. For communication between components of different modules, most components facilitate peer-to-peer communication, which is instantiated on demand (*e.g.*, input redirection or window migration). Each component can be re-configured at runtime to react to changes in the environment. If modules are added or removed, or if the user triggers a re-calibration (*e.g.*, after a display changed location), the master module updates the spatial model and re-configures individual components.

Figure 3.2 illustrates the basic software infrastructure of the Deskotheque environment with a single module. The three most important software components are shown: window management, mouse pointer navigation, and window migration. In the following subsections, these three components will be presented in more detail.

[*] `http://www.zeroc.com/`

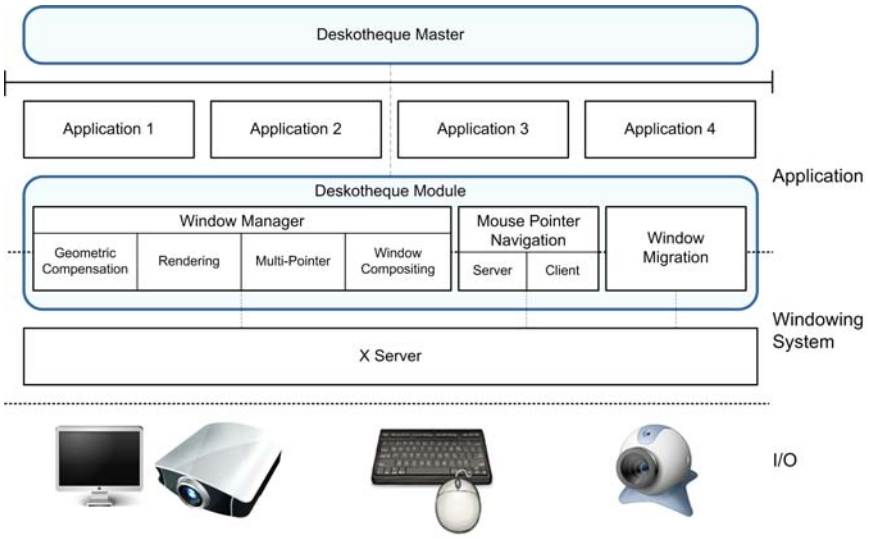


Figure 3.2: The central Deskotheque master and a single module with different software components, as well as input / output resources. Applications do not require any modifications to be operated in the Deskotheque environment.

3.2.1 Window Management

The window management component is operated by the Compiz window manager (Section 2.3.1) and is extended by multiple plug-ins. Despite this integration in the window manager, this component is not only responsible for window management, but also for information presentation tasks on the entire desktop imagery, as a compositing window manager provides convenient access to the resulting desktop imagery.

The four most important window manager extensions are illustrated in Figure 3.2. *Geometric compensation* takes care that the resulting desktop image is undistorted and uniformly blended when employing irregular or tiled projected displays (Section 7.3.2). A *rendering* extension draws arbitrary information on top of the desktop. Examples include structured light patterns (Section 4.1.1), visual navigation aids (Chapter 5), and visual links (Section 8.1). The *multi-pointer* extension renders multiple color-coded mouse cursors on the screen and provides fundamental multi-pointer support on the window manager level (Section 7.1). Finally, the *window compositing* extension is responsible for importance-driven compositing window management (Section 7.4) and all its associated window manager functions (*e.g.*, *Uncovering Windows* in Section 8.2 or *Display-Adaptive Window Management* in Section 8.3).

A networking extension serves as a communication interface between all plug-ins and the Deskotheque module and master, respectively. It establishes a connection with the Deskotheque environment using the *Ice* internet communication engine. A C++ wrapper of the Ice interface receives commands from the Deskotheque environment, translates the commands into a simple, Compiz-internal message format, and forwards it to the respective

plug-in(s).

The window management component is not dependent on any other component and therefore allows for stand-alone settings without any input redirection or application relocation support.

3.2.2 Input Redirection

Input redirection support is based on Synergy+MPX [71], which extends the open-source mouse pointer sharing tool Synergy [212] by facilitating MPX [115] for multi-pointer support (*cf.*, Section 2.3.2). An earlier multi-pointer input redirection solution in the Deskotheque environment used a custom multi-pointer control before MPX was available in standard Linux distributions [181], but will not be discussed in this thesis.

In the Deskotheque environment, for each machine with at least one connected input device (*i.e.*, mouse), a Synergy server instance is launched at start-up. On each host accessible for a respective pointer (see user and privacy management issues in Section 4.3), a corresponding Synergy client is launched, so the locally attached mouse input can be redirected to all accessible hosts. Each pointing device is assigned its individual navigation framework, using its own navigation coordinate system and navigation parameters, as described in Chapter 5.

At runtime, the pointer's associated navigation framework is consulted for each registered movement event. In the case of input redirection, *i.e.*, if the pointer is about to be redirected to an adjacent display and host, respectively, we grab the input events delivered by the pointer's associated *master device* (the software pointer rendered to the screen) and forward these events to the new client host. At the client host, the input events are received from the Synergy server, a new master device is created, and the input events are forwarded to this master device using the *XTest* extension[†]. The initial mouse pointer position is set according to the outcome position determined by the navigation framework. If the pointer leaves the client host again, its newly created master device is deleted.

Deskotheque keeps track of the individual pointers by tracking their master pointer IDs on the individual hosts. If an input redirection occurs, the Synergy server informs the Deskotheque master about the pointer's current host and its remote master pointer ID. Deskotheque then queries the uniquely assigned pointer's color and forwards it to Compiz' *multi-pointer* plug-in, which was introduced in the previous section. The mouse pointer plug-in identifies the pointer based on its current master pointer ID and renders its cursor with the correct color assigned. Figure 3.3 illustrates the involved components.

Mind that the *XFixes* extension[‡], which is required to query a pointer's currently assigned cursor, is limited to a single *core* pointer. Thus, we can only reliably query the current cursor image of one pointer, while all other pointers will be assigned the same

[†]`http://www.xfree86.org/current/xtest.html`
[‡]`http://freedesktop.org/wiki/Software/FixesExt`

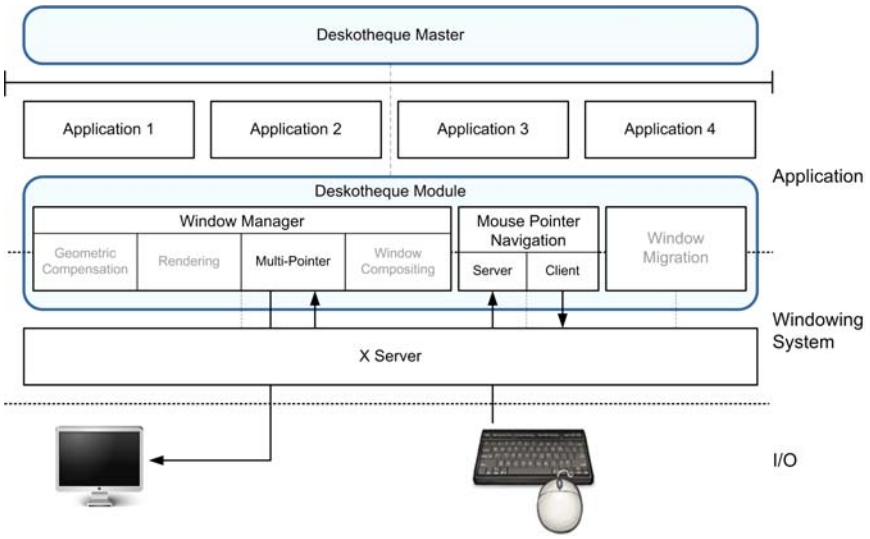


Figure 3.3: The mouse pointer navigation server listens for input events and, if required, modifies and redirects them to a remote client. Each redirection is forwarded to the master, which keeps track of each pointer's location in the environment. The multi-pointer plug-in is responsible to render the individual pointers in the correct color.

cursor image. Therefore, we currently only support the conventional arrow cursor. Early user feedback has indicated that a static cursor causes less confusion than a constantly changing cursor.

Synergy+MPX is not only facilitated in a distributed MDE. Even if Deskotheque is operated by a single machine, one Synergy server is launched which is responsible for mouse pointer movement corrections and transitions to adjacent, discontinuous displays.

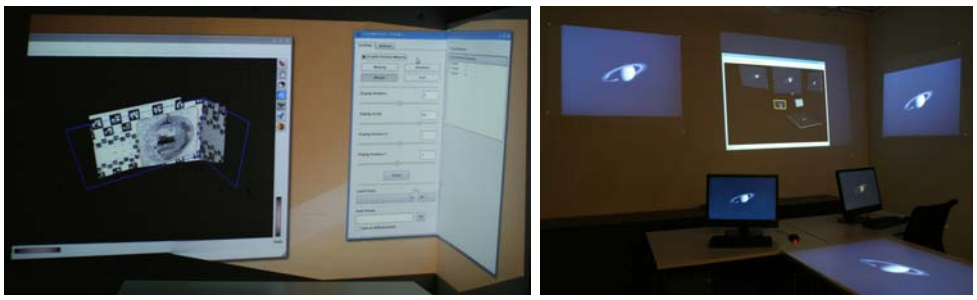
3.2.3 Window Migration

In an earlier version of Deskotheque [181], a window migration operation using *XMove* [221] was initiated when the window manager detected a window drag across a display border, or if a remote host was selected from a drop-down menu, integrated into the window's title bar. The input redirection component then determined the mouse pointer's outcome position on the remote display and informed the window migration component about the window's remote host and outcome position.

However, XMove was found to be unreliable and unstable and was therefore removed from the Deskotheque framework. In the current version, window relocation is only possible within single-machine setups, where the window manager implicitly supports drag-and-drop across display boundaries. In the future, novel approaches in the X Window System (see Section 2.2.4) could solve the problem of window migration more reliably.

3.2.4 Other Components

As these three core components are tightly integrated into the X Window system, legacy applications can be operated in the environment without any modifications. However, special-purpose applications or modified applications can also facilitate the existing network interface. They can exploit the framework's knowledge of the physical environment (*i.e.*, display arrangements and user locations), request user identities to received input events, and instruct the framework to render information on top of the desktop. Two examples are shown in Figure 3.4.



(a) (b)

Figure 3.4: Applications facilitating the Deskotheque interface: (a) The display control center (right window) allows for interactive scaling and rotation of the desktop within the display outline. (b) The distributed world-in-miniature control requests all current desktop images and the environment reconstruction to provide an interactive pointer control (Section 5.3.3).

3.3 Discussion

The Deskotheque infrastructure supports two common approaches of middleware support: First, it allows for legacy application operation in the MDE. Fundamental operations, like undistorted projected imagery, input redirection in a distributed system, and window migration to remote hosts or displays, are provided by the underlying framework, transparent to any X-compatible application. Legacy application support increases applicability of the MDE and encourages a wide-spread adoption. It has to be noted, however, that window migration support using *XMove* [221] is limited to very basic applications and often fails if recent X extensions are utilized by the application to be migrated.

Second, the infrastructure provides a networked API so special-purpose applications can be tailored towards the environment. They can query information about the environment, such as display form factors or user arrangements, and request certain operations, such as rendering information across display boundaries. Special-purpose applications can co-exist with unmodified legacy applications in the same session.

The restriction of the multi-display framework to the X Window System has been a careful design decision. Linux allows for easier access to low-level features compared to

other platforms. For instance, window migration solutions for Linux are not wide-spread and far advanced, but basic implementations and research prototypes are available, in contrast to other platforms (*cf.*, Section 2.2.4). Another important aspect was the flexible plug-in infrastructure of our employed compositing window manager (*cf.*, Section 2.3.1), which allowed for efficient access to individual window textures, or the entire desktop image, as well as for easy modifications of the window rendering procedure or rendering of additional information on top of the desktop imagery. Although most platforms now rely on 3D compositing window managers, the access on other platforms is usually restricted to very few API calls.

The Deskotheque infrastructure has never been considered as "finished" or close to finished. As for most low-level software engineering, providing such a basic infrastructure is facing the "moving target" problem [161], *i.e.*, the underlying software infrastructure fundamentally changes during development and makes certain components inoperable or obsolete before even finishing a first research prototype. Therefore, and for the sake of reduced maintenance, some of the interface prototypes and interaction techniques presented in this thesis do not operate in the distributed system, but are rather limited to single-machine MDEs.

Chapter 4

Spatial Awareness in the Deskotheque Environment

Spatial awareness of the display arrangements, individual display form factors, and user locations towards the displays, is the most distinguishing aspect of the Deskotheque environment compared to other MDE infrastructures. A fine-grained model of the environment is acquired in an offline calibration process. The calibration process is based on a set of standard algorithms for structured light and multi-view reconstruction. The environment is observed by at least two cameras with known internal parameters to create a 3D model of the displays.

The calibration procedure is separated into the following steps:

1. **Internal camera calibration:** the intrinsic parameters of the cameras used for calibration have to be known a priori. The internal camera parameters can be retrieved with standard tools, like the well-known camera calibration toolkit for MATLAB by Bouguet [46] or OpenCV [47].

2. **Spatial model creation:** the spatial model of the display environment is created using the internally calibrated cameras and sequential structured light patterns for projectors and monitors. This process is explained in detail in Section 4.1.

3. **Environment configuration:** based on the spatial model, environment configurations, such as geometric compensation and blending of tiled projected displays, user location estimation, and mouse pointer navigation frames, are estimated. These configurations are discussed in sections 4.2, 4.3, and 4.4.

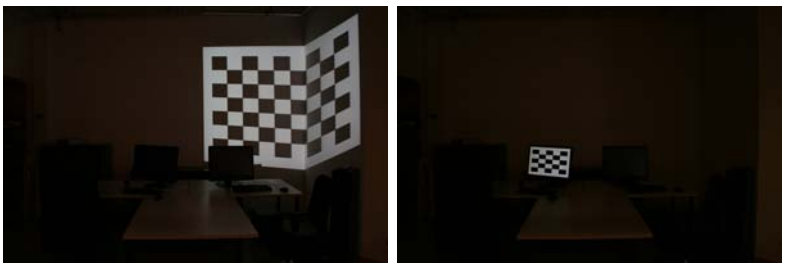
At runtime, the system will facilitate the environment configurations for spatially consistent interaction and visually adjusted content (Parts III and IV). There will be no user tracking or constant observation of display changes in the environment, as we assume an office environment to be only partially dynamic. The system was designed to allow for (partial) re-calibration on demand (*e.g.*, if a user changed location of her laptop) and to accommodate for late incomers.

4.1 Spatial Model Creation

The spatial model creation step is an offline calibration procedure facilitating at least two calibrated cameras and sequentially displayed structured light patterns (Section 4.1.1). From the detected structured light feature points, a sparse point cloud reconstruction is created (Section 4.1.2). As we assume that projected displays are limited to planar or multi-planar surfaces, we derive a polygonal model from the point cloud per output device (Section 4.1.3). Overlapping projections are automatically detected from the model. Each projection surface reconstruction is then parameterized as a 2D surface (Section 4.1.4), where the screen rectangle to be displayed is inscribed or circumscribed to derive a perfectly rectangular projected imagery (Section 4.1.5).

4.1.1 Structured Light

Structured light patterns are generated upon calibration start-up based on some user configuration for each output device. The user has to specify the type and parameters of the calibration pattern in a configuration file for each output device individually. In its first version, Deskotheque supported two different structured light patterns: checkerboards (based on OpenCV [47]) and binary Gray codes in combination with phase shifting (*cf.*, [209]).



(a) (b)

Figure 4.1: Checkerboard patterns for (a) an irregular projected display and (b) a monitor, captured by the same camera.

While the checkerboard detector by OpenCV is very accurate and fast for simple projector configurations, Gray code patterns can be employed to deal with non-planar or partially occluded projection surfaces. However, in practice, there are MDE configurations that cannot easily be handled with these standard procedures. As an MDE may be composed of heterogeneous display devices, there may be considerable differences in display illumination, size, and resolution (*cf.*, Figure 4.1). Especially the low brightness and size of monitors in comparison to the projectors was found to cause detection problems, and was additionally impeded by oblique camera-monitor angles and the resulting brightness loss. To overcome these situations, the user was required to manually set a camera exposure

that could handle displays of strong brightness differences and manually configure the structured light parameters for the different display sizes in the camera image.

As a solution, an iterative structured light process, tailored to MDEs with multi-planar projections and monitors, was introduced [160]. The structured light technique facilitates two properties to overcome heterogeneity in multi-display configurations:

1. An increasing number of fiducials markers (based on *ARToolKitPlus* [252]) is iteratively displayed on each output device (Figure 4.2(a)). Starting with a single (almost) screen-filling marker, one row or column of markers, respectively, is added until some stop criteria is fulfilled (*e.g.*, a given percentage of markers or all markers were detected, or the maximum number of iterations was reached). For accurate subpixel detection, markers are substituted with checkerboards in each iteration. This step assures that size differences can be compensated while multi-planar projections can still be detected in sufficient quality.

2. For each displayed pattern, images are taken with different exposure values and composed into a high-dynamic range image with minimized noise (Figure 4.2(b)). Producing high-dynamic range camera images leads to an increased calibration time, but differences in brightness between displays can be effectively compensated.

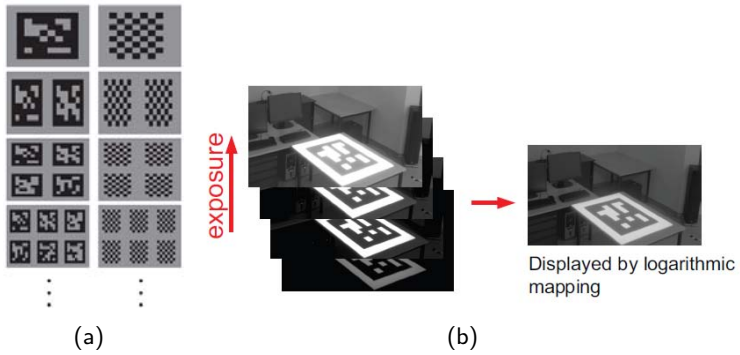


Figure 4.2: Iterative marker-based structured light process (from [160]): (a) sequence of projected images and (b) sequence of captured camera images with different exposure, combined into a high-dynamic range image.

4.1.2 Point Cloud Acquisition

Structured light patterns are displayed sequentially for each output device and are captured by each camera. The detectable structured light feature points in screen coordinates, as well as the actually detected feature points in each camera's coordinate system, are sent to the master process. Detected pattern points deliver point correspondences between camera images. From all available cameras, the two cameras with most common point

correspondences (irrespective from which displays they were obtained), represent the base stereo camera pair.

As first step, the relative camera poses of the base camera pair, *i.e.*, the rotation and translation of the two cameras towards each other, are estimated. Deskotheque uses the computer vision library *VRLib** for multi-view geometry operations. For relative pose estimation, the Five-Point relative pose algorithm by Nistér [172] is used, which estimates the essential matrix from five point correspondences using a RANSAC [76] approach (`VROrientation::computeRobustOrientation`). The results yield the rotation R and translation t of the second camera towards the first camera. The first camera is set to the center of the environment's world coordinate system. As the external camera parameters of the base camera pair are now known, as well as the internal camera matrices K_i, the projection matrix of the two cameras can now be set to

$$P_1 = K_1[I|0]$$

and

$$P_2 = K_2[R|t].$$

As second step, a point cloud reconstruction of the feature points detected by the base stereo pair is obtained by triangulation of each point correspondence (`VRTriangulatedPoint::reconstruct`), resulting in a sparse model of each output device. The density of the point cloud depends on the employed structured light technique.

In an iterative procedure, the absolute poses of the remaining cameras are estimated from the already reconstructed feature points. To estimate rotation R and translation t for a calibrated camera from known world coordinates (absolute pose estimation), at least three 2D-3D point correspondences are required. Again, the VRLib implementation uses RANSAC for a robust estimation (`VROrientation::computeRobustAbsolutePose`). The reconstruction is then updated for feature points detected by the newly added camera and the cameras with already known projection matrix.

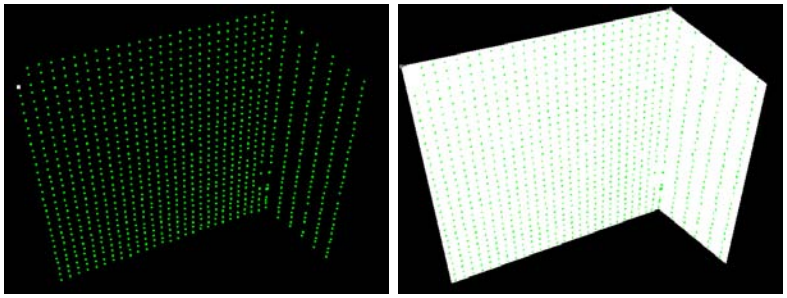
The final reconstruction is refined by a bundle adjustment routine, optimizing camera poses and 3D points, but leaving the internal parameters unmodified (`VRBundle::adjustStructureAndMotion`).

4.1.3 Polygonal Model Creation

As Deskotheque is limited to planar or multi-planar displays, the established point cloud can be approximated by a polygonal model. As depicted in Figure 4.3(a), the resulting point cloud may consist of a large number of vertices (approximately 500 in this example) and be subject to reconstruction noise, due to detection or reconstruction inaccuracies. In contrast, fitting a polygonal model into the point cloud dramatically reduces the number

*Developed by the VRVis "Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH" Vienna (`http://www.vrvis.at/`)

of required vertices (6 in the example of Figure 4.3(b)), while the physical corners are properly modeled.

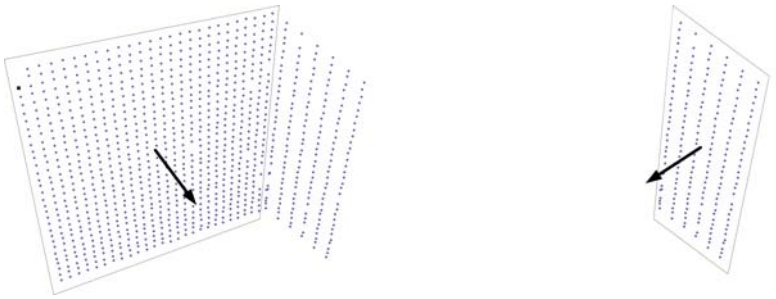


(a) (b)

Figure 4.3: Reconstruction of a multi-planar projection surface: (a) The point cloud reconstruction and (b) the approximated polygonal model .

Plane Fitting

An iterative RANSAC-based plane fitting algorithm is applied to each output device's obtained point cloud. The aim is to approximate piecewise planar display areas, as proposed by Quirk *et al.* [187]. As illustrated in Figure 4.4, the algorithm iteratively finds the plane containing the highest number of inliers (`VRPlane::ransac`) and proceeds with the remaining outliers.



(a) (b)

Figure 4.4: Iterative plane-fitting procedure.

Polygon Outlines

The fitted planes are infinite and should be delimited by the physical output device boundary. The device boundaries themselves are normally not captured, as structured light patterns are often limited to the interior areas of the screen, while keeping a white boundary (see Figure 4.1 or 4.2). To determine the vertices of the output device polygons, we need

to determine the four corner vertices of the output device in world coordinates, as well as the edge pixels of potential plane intersections in world and screen coordinates.

Each displayed structured light point can be associated with its 3D reconstruction in world coordinates and a fitted plane i. Thus, for each fitted plane i, there is a screen→plane homography $H_{\pi i}$ relating the output device coordinates to the plane coordinates of plane i. To find all possible world coordinates of the four output device corners, we map the corners to all fitted planes, using the screen→plane homographies $H_{\pi i}$ (Figure 4.5(a)). Each polygon is now limited by the mapped output device corners. If only a single plane has been fitted, the polygonal model is now fully defined.

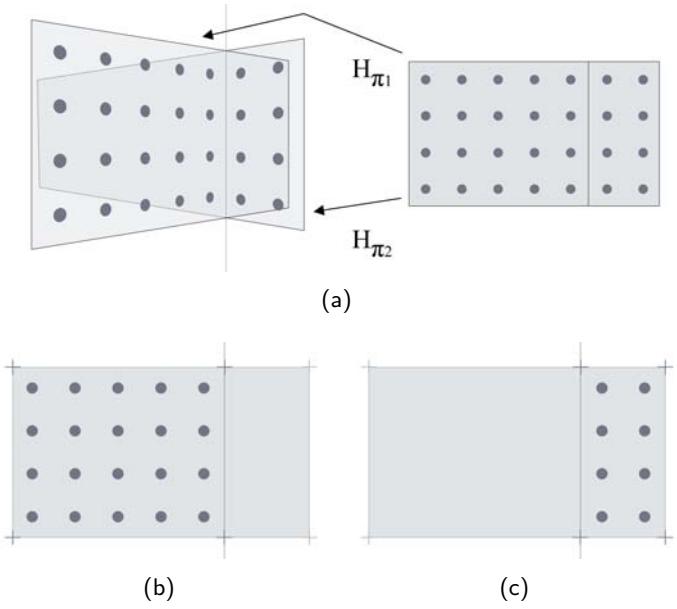


Figure 4.5: Polygonal model creation: (a) The output device outline is mapped to each world plane and the two planes are intersected. (b,c) The intersection line is mapped to output device coordinates and is used to refine the extents of the two output device polygons.

To find the pixels along physical corners, we intersect each fitted plane pair i and j in world coordinates and map the resulting line to output device coordinates, using the inverse screen→plane homographies $H_{\pi i}^{-1}$ and $H_{\pi j}^{-1}$. For both polygons, the line is intersected with each polygon edge (Figure 4.5(b) and (c)). In addition, the centroid of structured light feature points associated with the fitted plane is projected onto each polygon edge. If a valid intersection between the plane intersection line and a polygon edge was found, the intersection point replaces the polygon vertex, which lies on the edge's same side with respect to the mapped centroid. For instance, on the top edge of the polygon in Figure 4.5(b), the intersection point lies on the right side of the projected feature point centroid and therefore replaces the right output device corner point. The obtained polygon vertices can now be mapped back to world coordinates, and serve as new polygon outline for any further plane intersection refinements, if necessary.

Detecting Overlapping Projections

From the individual polygons, overlapping projections can be automatically detected. For each possible polygon pair, the polygons are checked for co-planarity and, if co-planar, for edge intersections on the common plane. If two output devices' polygons overlap, they are treated as a continuous, tiled display.

As a result of the polygonal model creation step, all displays are approximated by a single or multiple polygons and registered into a common, metric coordinate system. The origin of this coordinate system is defined by the first camera of the initial stereo camera pair, and is usually adjusted slightly by the bundle adjustment step (Section 4.1.2). Figure 4.6 shows an example polygonal model for an office environment.

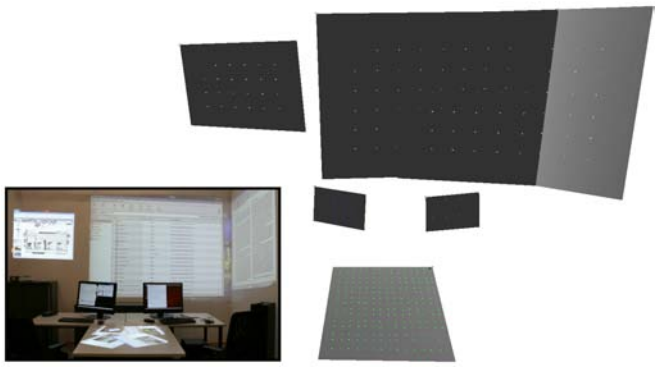


Figure 4.6: The resulting polygonal model for an office environment: the colored dots correspond to the detected structured light feature points (color-coded per output device). Overlapping projections are treated as a single display (right wall display).

For monitors, the offline calibration procedure is finished with the polygonal model creation step, as no geometric compensation and brightness adjustment is required. Using the screen→plane homographies, each monitor pixel can be mapped to a 3D location in the environment. For projected displays, some additional calibration steps are required, to calculate geometric compensation and alpha blending of overlapping projections, as discussed in the following sections.

4.1.4 Surface Parameterization

The aim of a geometric correction routine for projected displays is to map an existing 2D rectangle to a 3D polygon with minimal geometric distortion. In the case of the Deskotheque environment, the 2D rectangle is represented by the X Window System's screen and the 3D polygon represents the reconstructed projection surface.

Different strategies to compensate for distortion, including view-dependent and view-independent approaches, were discussed in the background chapter (Section 2.2.2). As Deskotheque was designed with multiple users in mind, we rely on a view-independent method, which aims for a wallpaper-like effect, where the image looks undistorted when

viewed from a location perpendicular to the surface. Of course, wallpaper-like compensation is only possible, if the projection surface is developable, *i.e.*, the surface can be "flattened" without any seams or deformations.

Parameterization Techniques

In the Deskotheque system, two surface parameterization methods are currently supported, which differ in the way potential non-developable surfaces are handled:

Least squares conformal maps [141] were originally introduced as a texture mapping technique, but have been successfully employed for view-independent geometric compensation for projector-based displays [190]. If the projection surface is non-developable, the algorithm aims at minimal distortion, but leads to deformations (Figure 4.7(a) and (d)).

The **patch parameterizer** maps individual polygons to a common 2D surface without distortion, but instead introduces seams. It aims to preserve vertical display alignments, therefore keeping vertical edges intact while introducing seams along horizontal edges, if the surface is non-developable (Figure 4.7(b), (e)). As the seams introduce "holes" within the projected imagery (Figure 4.7(f)), the runtime component has to take care that window placement and mouse pointer navigation is seamlessly possible.

Alignment in Texture Space

Deskotheque aligns the parameterized surface in texture space, either along the averaged vertical display outlines outlines (*i.e.*, left-most and right-most polygon edges) or along averaged physical vertical corners, if available. In the example of Figure 4.7, the display is aligned along the single vertical edge. Polygon edges are assumed to be vertical, if the mapping of the edge in screen space is approximately vertical (*cf.*, Figure 4.7(c)). Deriving a common world vertical for the entire environment from the world coordinate system, defined by the first camera of the base stereo camera pair, is too unreliable, as cameras are often mounted in a tilted fashion. For a reliable global world vertical, an additional physical marker would be required (*e.g.*, a printed fiducial marker).

Virtual Display Creation

In Deskotheque, multiple projected images can be composed to a seamlessly tiled display. As described in Section 4.1.3, projection overlaps are automatically detected and all contributing output devices are treated as a common, continuous display. Each of these virtual displays has its own display coordinate system, *i.e.*, a rectangular coordinate system ranging from $(0, 0)$ to $(N_h width, N_v height)$, where N_h is the number of horizontal output devices and N_v the number of vertical output devices, arranged in a rectangular fashion. All output devices are assumed to have the same number of pixels, *i.e.*, $width$ and $height$ are equal across all projectors.

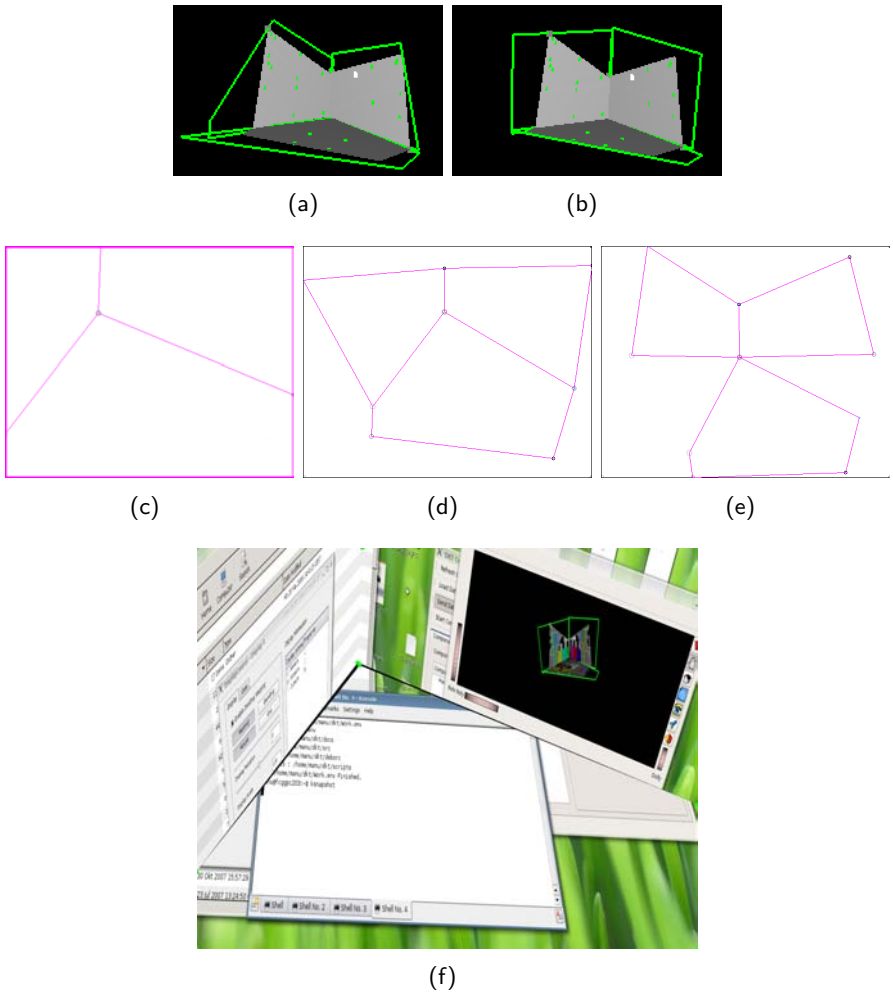


Figure 4.7: Surface parameterization for non-developable surfaces: The 3D polygonal model with virtual display outlines obtained from (a) LSCM and (b) the patch parameterizer. Mind that the reconstruction and resulting polygonal model is imperfect, thus polygon corners do not perfectly align in world coordinates. The polygonal model in 2D coordinates: (c) screen coordinates, (d) LSCM parameterization, and (e) patch parameterization. (f) A screenshot of the geometrically compensated display using the patch parameterizer. Portions of the desktop are invisible due to the seams.

As displays are not necessarily composed of projectors connected to one single machine, Deskotheque has to determine N_v and N_h automatically to calculate the resulting display rectangle. Deskotheque supports two modes to calculate the spatial display layout: In the "manual" mode, each output device's row and column within the display rectangle is derived from its X Window System's screen coordinates. Thus, each output device's *screen coordinates* ($\mathbf{x_s}$), as defined by the X Window System, are identical to Deskotheque's virtual *display coordinates* ($\mathbf{x_d}$). The user selects this mode if each continuous display corresponds to one machine and the physical display topology corresponds

to the X Window System's screen layout.

In the automatic mode, the spatial relationship of output devices towards each other is derived from the 2D mapping of the output devices in texture space. The row and column of an output device is determined by comparing its center towards the diagonals connecting the outer corners of the closest output device with already known row and column. This way, displays with diverging X Window System coordinates (*e.g.*, driven by different machines) can be constructed. As we assume the same number of pixels per output device, the relation between $\mathbf{x_s}$ and $\mathbf{x_d}$ can be described by a simple translation.

4.1.5 Display Rectification

The rectification step finally determines how the display rectangle is placed within the texture space containing the parameterized projection surface. A popular approach is to inscribe the rectangle within the potentially concave outline of the display [190]. This assures that all image pixels are visible on the surface. However, if the display outline's aspect ratio strongly deviates from the image rectangle, or if it has strong concavities, this approach leads to a significant minification of the displayed imagery and, as a result, interpolation artifacts.

In the Deskotheque environment, we therefore provide the possibility to circumscribe the display rectangle around the parameterized display outline (*cf.*, Figure 4.7(d) and (e)). This leads to a loss of information at the display boundaries, or along potential seams (*e.g.*, Figure 4.7(f)). However, display-aware content management at runtime (*cf.*, *display-adaptive window management* introduced in 8.3) helps users to optimally use the available space and explicitly makes use of irregularities introduced by non-rectangular and non-planar projection surfaces. In addition, we allow the user to adjust the rectangle's placement and size interactively at runtime (*cf.*, Figure 3.4(a)).

4.1.6 Results

The duration of the calibration process is mainly dependent on the chosen structured light technique. While a simple checkerboard can be detected in a less than a second by two cameras, iterative marker-based patterns in combination with HDR camera images [160] may take up to several minutes for a single display to get detected, as a large number of camera images needs to be captured. It has to be noted, though, that cameras and display devices are not synchronized in the Deskotheque environment. Although we try to estimate the capturing duration of the cameras in a pre-calibration step, this value is only a rough estimate and not fully reliable.

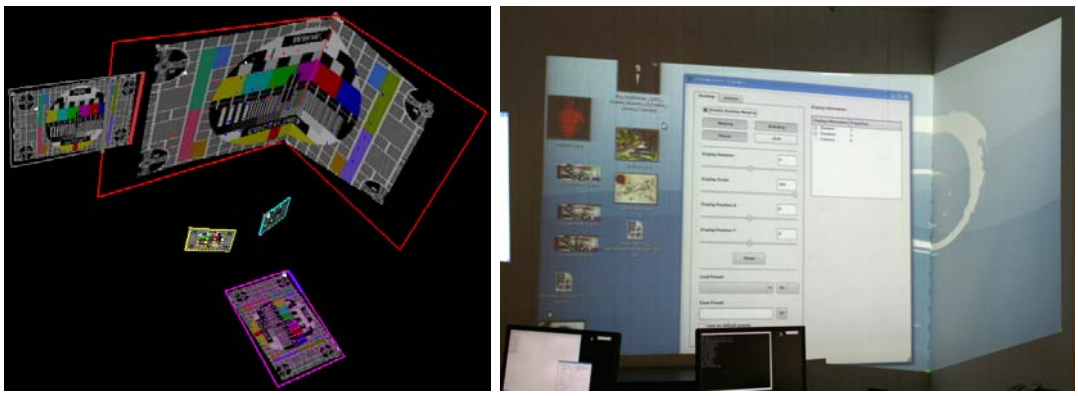
The accuracy of the spatial model depends on several parameters:

- the accuracy of the internal camera calibration and camera lens distortion estimation,
- the stereo baseline (*i.e.*, the distance between two cameras),

- the size of the environment and, as a result, the size of the individual displays in the camera images,
- the viewing angle of the camera(s) towards a screen,
- the amount of lens distortion of the projectors, which is not compensated,
- and the sharpness of the projected imagery.

In addition, we observed that incorrect assignments of structured light feature points close to a physical corner to a randomly set plane in the polygonal model creation step (Section 4.1.3) lead to slight inaccuracies. This problem was observed in particular if structured light feature points were located very close to physical corners, such as in Figure 4.1(a). The iterative marker-based approach of [160] helped to work around this limitation, as fiducial markers are only detected when entirely located within a planar sub-region of the screen, therefore keeping structured light feature points from being too close to physical corners (Section 4.1.1).

Figure 4.8(a) shows the result of the surface parameterization and display rectification steps for the model of Figure 4.6. The textured quads show the reconstructed output device polygons and the color-coded outlines indicate the rectified virtual display. We used the smallest circumscribed rectangle as rectification method. As the overlap between the two projectors on the right wall display is large, this results in a significant loss of pixels, mainly at the lateral screen regions. The dashed line along the estimated polygon edge in screen coordinates in Figure 4.8(b) illustrates the accuracy of the plane fitting result with respect to the real world.

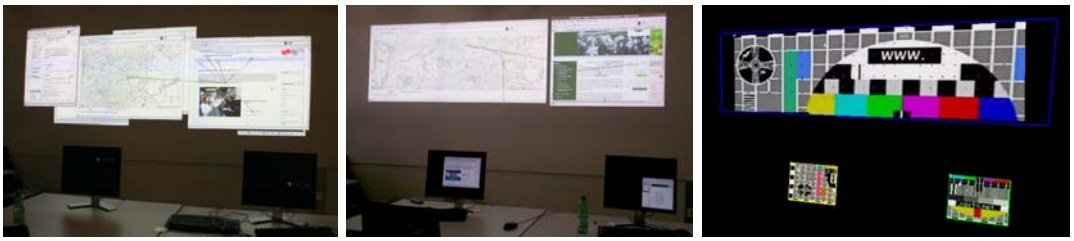


(a) (b)

Figure 4.8: (a) Result of the surface parameterization and display rectification process on the polygonal model of Figure 4.6. (b) The close-up to the multi-planar, tiled display shows a slight inaccuracy of the polygon edge along the physical corner (display warped and blended).

In the example of Figure 4.8, each display is driven by its own host, *i.e.*, the virtual display coordinates correspond to the X Window System's screen coordinates. Each com-

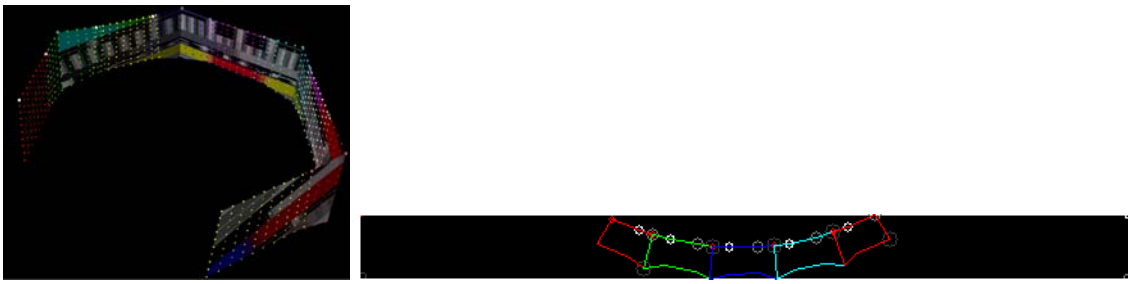
pound X Window screen is visualized by a test image in the reconstruction of Figure 4.8(a). In contrast, Figure 4.9 shows a setup driven by a single machine, facilitating VGA splitters[†]. In Figure 4.9(a), the six projectors correspond to the X Window System's virtual output device arrangement. However, in Figure 4.9(b), the single X Window screen is distributed among the tiled projected display and the two discontinuous monitors. In a configuration file, the user has to assign individual portions of the X Window screen to the respective output devices, but the virtual display coordinates of the projected display are detected automatically during the calibration process.



(a) (b) (c)

Figure 4.9: Tiled displays composed by single host machines: (a) six projectors and (b) three projectors, in combination with two monitors. (c) The reconstruction result illustrates how the X screen pixels are assigned to the individual displays.

Figure 4.10 illustrates an example of a developable surface, where both surface parameterization methods provided in the Deskotheque framework lead to an unsatisfactory result. Due to the slightly suspended corners of the surround projection, the parameterization leads to a space-wasting mapping in texture space and the impression of a curved image on the display (Figure 4.10(b)). Therefore, in an extension of the Deskotheque system, which is not part of this thesis, Pirchheim *et al.* [180] map the surround projection onto a centered, vertical cylinder, which is then unfolded into texture space.



(a) (b)

Figure 4.10: A developable surface with slightly tilted vertical edges: (a) From the developable (simulation) point cloud, (b) the patch or LSCM parameterizers create a curved mapping in the texture space.

[†]Matrox TripleHead2Go: `http://www.matrox.com/graphics/en/products/gxm/th2go/`

4.2 Compensation of Projected Displays

The creation of multi-projector displays in the Deskotheque environment allows for a visually seamless presentation space across multiple, casually aligned projections. The projector calibration step calculates geometric warping of projected images with respect to a display rectangle, and blending of multiple overlapping projections to compensate for uneven brightness. It does not apply any photometric correction, *i.e.*, compensating for color variances between multiple projectors contributing to the same display.

At runtime, geometric compensation and blending is applied by a window manager extension (Section 7.3.2). Warping is only calculated for projected displays, blending for tiled projected displays.

4.2.1 Warping

For geometric compensation (or warping) of projected imagery, Deskotheque relies on an approach frequently employed for planar tiled displays (*cf.*, Section 2.2.2): 2D homographies are used to perspectively correct each projector's imagery with respect to a rectification in a common 2D coordinate system (usually a camera image).

In contrast to planar displays, homographies in the Deskotheque environment are required for each planar sub-region of an output device. In addition, the planar coordinate frame describing the perspectively distorted projections and projection overlaps, respectively, is not defined by a camera image, but by the texture space describing the parameterization of the projection surface (Section 4.1.4), to ensure view-independence.

Each output device is described as at least one polygon, depending on the planarity of the projection surface. Homographies for warping are defined for each polygon and are composed from three different mappings:

- H_{id} maps the screen coordinates of a projector $i = 1, .., N$ (Figure 4.11(b)) to their corresponding virtual display coordinates (Figure 4.11(c)). If the display rectangle's topology corresponds to the projectors' X Window System's screen coordinates, this matrix is identity. Otherwise, it describes a simple translation.

- H_{du} maps the entire virtual display rectangle to its rectification in texture space (Figure 4.11(d)). This mapping describes a translation and scaling.

- H_{uj}, finally, maps each polygon $j = i, .., M$ (where $M \geq N$) from texture space to screen coordinates. This mapping is a perspective transformation in $\mathbb{R}^2$.

The resulting perspective transformation is composed by:

$$H_j = H_{uj} H_{du} H_{id}.$$

The warping function is applied for each polygon j and maps screen coordinates ($\mathbf{x_j}$) to

undistorted screen coordinates ($\mathbf{x}'_\mathbf{j}$):

$$\mathbf{x}'_\mathbf{j} \sim H_j \mathbf{x}_\mathbf{j}.$$

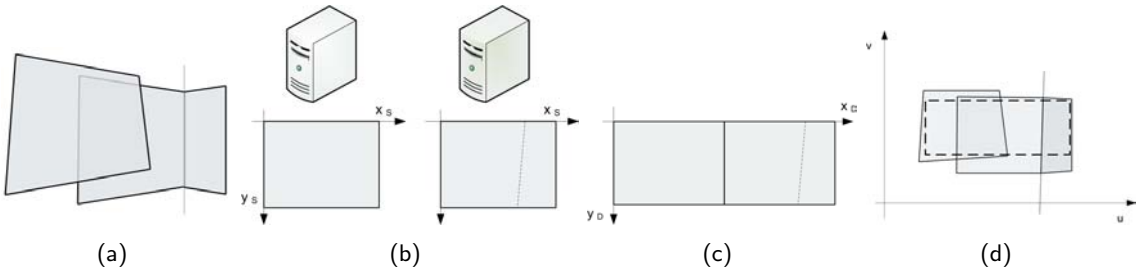


Figure 4.11: Coordinate systems for geometric compensation: (a) The world coordinate system consisting of two projected displays with one display spanning across a physical edge, (b) the screen coordinate systems of the two projectors, which are assumed to be operated by two PCs, (c) the acquired display coordinate system, and (d) the mapping of the world coordinates to a 2D texture space with inscribed display rectangle.

As described in more detail in Section 7.3.2, the runtime component renders each texture-mapped polygon j in screen coordinates, and applies the associated geometric transformation H_j.

4.2.2 Blending

To preserve approximately uniform image brightness across overlap regions of adjacent projected images, intensity blending with linear ramps was proposed by Raskar *et al.* [191]. An appropriate alpha blending value is calculated for each pixel in each projector's normalized (*i.e.*, [0...1]) screen coordinate system. For each screen pixel $\mathbf{x}$ of a projector m, the alpha value $\alpha_m(\mathbf{x})$ (in the range [0...1]) is calculated as follows (adapted from [191]):

$$\alpha_m(\mathbf{x}) = \frac{d_m(\mathbf{x})}{\sum_i d_i(H_{uj} H_{uk}^{-1} \mathbf{x})},$$

where $d_m(\mathbf{x})$ is the minimum distance of $\mathbf{x}$ from the output device border in projector m and d_i the distance from the border in all other projectors $i = 1, .., N$. H_{uk}^{-1} maps $\mathbf{x}$ to the texture coordinates of its associated polygon k, while H_{uj} maps it to the screen coordinates of polygon j in projector i.

The resulting alpha values are stored in the alpha channel of a texture for each projector. As calculating the alpha value for each pixel is computationally expensive, the default size of the texture is 512x512.

4.3 User Location Estimation

From the spatial model of the environment, we can estimate the location of the individual users by exploiting some simple assumptions. First, we assume that each user operates the environment with a single mouse and keyboard pair. We can furthermore assume a user's "home" display, connected to the same PC as the input devices. From the associated home display, we can then estimate the user's head location to be at a certain distance from the home display's center, with a viewing direction perpendicular to the display surface, as illustrated in Figure 4.12.

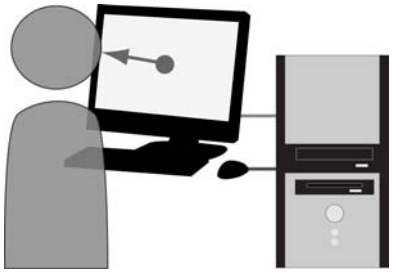


Figure 4.12: The user's head location is estimated from the associated home display.

Most likely, mouse and keyboard will be connected to a personal workstation with one or two monitors, or a laptop. Using this assumption, we can also reasonably assign pointing devices to single-machine environments. For instance, in the single-machine MDE of Figure 4.9(b), the two mouse and keyboard pairs on the table are associated with the two monitors. The projected display is automatically handled as public space without explicitly associated input device.

The concept of home displays introduces an implicit level of privacy as comparably small monitors facing away from co-workers ensure a certain intimacy. This privacy should not be interrupted by "intruding" mouse pointers. By default, monitors with connected mouse pointers are treated as private display spaces, accessible only for the host mouse pointer. Projection walls are public displays open to any user.

These assumptions work well in personal device-centric environments, such as conference rooms and open-plan offices, where the user is assumed to operate the MDE with her personal input devices and also has a personal home display. This approach does not require the users to wear additional hardware to determine their location, allowing for ad-hoc usage of the MDE and reducing overall cost. In addition, early experiences with unobtrusive user tracking[‡] in the Deskotheque environment [197, 231] have shown that tracking accuracy is low and suffers from significant jitter. For a more reliable result, also delivering information about the user's viewing direction, more obtrusive and expensive tracking equipment, such as 6-DOF infrared tracking[§], would be required.

[‡]Ubisense Ultra-Wide-Band Tracking `http://www.ubisense.net/en/`

[§]*e.g.*, `http://www.ar-tracking.de/Products.8.0.html`

4.4 Multi-Display Coordinate Systems

For several components in the Deskotheque environment, a world coordinate system of the environment is required. The 3D world coordinate system directly obtained from the calibration procedure is not always sufficient. An exception is the world-in-miniature control described in Section 5.3.3, which provides an interactive, live-textured 3D model of the environment by simply rendering the polygonal model in a 3D scene.

For multi-display navigation using indirect pointing devices (Part III) or cross-display visualizations (Section 8.1), we require a 2D mapping of the environment, so 2D paths can be uniquely mapped onto the environment. For user interface elements rendering and 2D input control, operating systems usually consider the individual output device's screen coordinate systems, defined by their number of pixels, and arrange them side-by-side (Figure 4.13(a)), irrespective of their resolution (Figure 4.13(b)), distance (Figure 4.13(c)), or their 3D arrangements (Figure 4.13(d)).

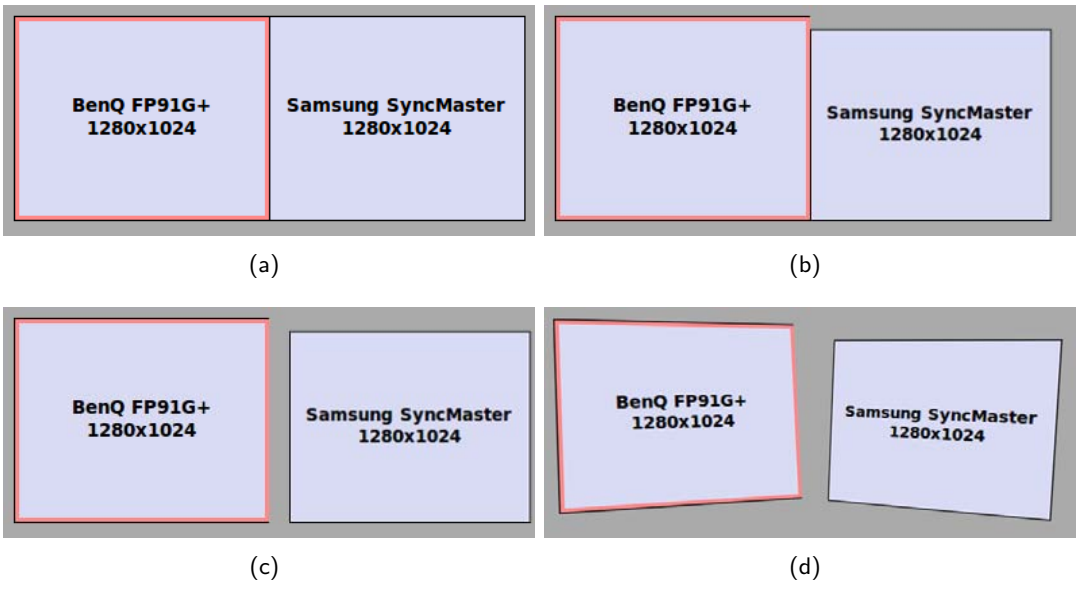


Figure 4.13: Possible spatial 2D mappings of a simple dual monitor setting: (a) the operating system's side-by-side arrangement of the individual screens, (b) additionally taking into account the resolution and (c) the distance between the monitors, or (d) the 3D reconstruction of the monitors, mapped to the user's perspective.

Nacenta *et al.* [166] group input models for MDEs into planar (Figure 4.13(a)-(c)) and perspective (Figure 4.13(d)). As additional category in their taxonomy, literal techniques require physical contact of direct pointing devices or adjacent displays (*e.g.*, *Pick and Drop* [193], *Stitching* [107], or *Connectables* [239]) to transfer information between displays. These techniques do not require a pre-defined multi-display coordinate system.

In complex multi-display environments, simple planar arrangements, such as shown in Figure 4.13 (a)-(c), cannot always appropriately capture all individual displays. Consider,

for instance, Figure 4.14: As the reconstruction is not developable, there is no intuitive 2D mapping of the environment without introducing distortions to the individual displays or irregular seams. In particular, the central wall display, the right wall display, and the tabletop display cannot be aligned without distortions or disruption of physical edge-to-edge alignments.

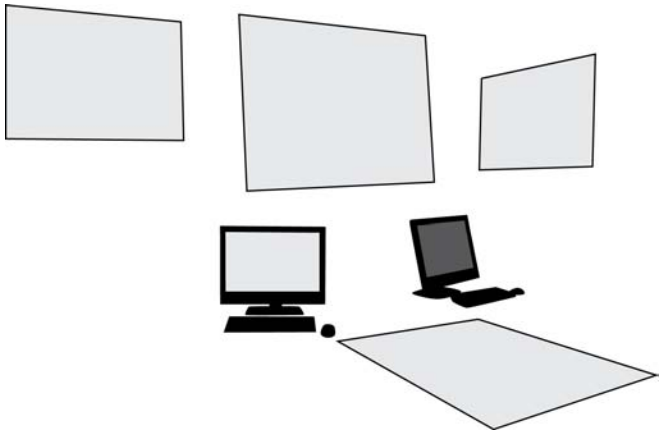


Figure 4.14: Example of a complex MDE without intuitive 2D mapping: the reconstruction is not developable.

Deskotheque provides two environment mapping techniques for complex MDEs: pair-wise planar mappings and perspective mappings. These two techniques will be described in more detail below.

4.4.1 Pair-Wise Planar Mappings

In this approach, the environment is segmented into multiple display pairs, each of which is mapped onto its own planar coordinate system, such as shown in Figure 4.13(b). As can be seen in the figure, the pair-wise planar mapping considers the displays' resolutions and their physical alignment, but display-less space between the displays is not considered in the mapping.

Display pairs are aligned along their closest (outer) edges, regarding their physical display-less space. The closest edges are derived from the three-dimensional representation of the workspace, for each possible display pair. Once the closest edges are found, the two corner points of the edges are projected onto the complimentary edge on the adjacent display, as illustrated in Figure 4.15(a). The respective edge interval, where the adjacent display is aligned, corresponds to the overlap of the display's edge and the projected complimentary edge. The resulting edge intervals are visualized in gray in Figure 4.15(a). If the resulting interval is empty or smaller than a given threshold, the candidate is discarded. Otherwise, the normal distance of the edges at the midpoint of the edge interval is used as a measure of proximity.

Overlapping edge intervals from different display pairs are prioritized according to

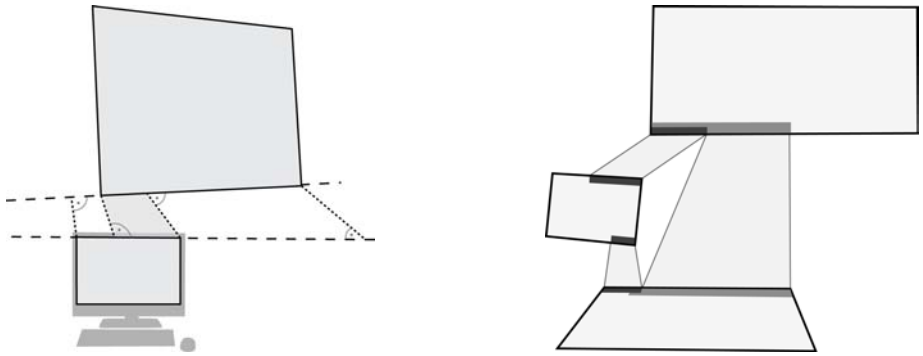


Figure 4.15: Finding pair-wise planar mappings: (a) The corner points of the closest edges of monitor and central wall projection in Figure 4.14 are projected onto each other. (b) Overlapping edge connections are prioritized according to their proximity score.

their proximities (Figure 4.15(b)). Edge intervals overlapped by intervals of higher proximity scores are trimmed or removed, if the remaining size is below the minimal interval threshold.

Proximity scores can be furthermore penalized, depending on the anticipated application scenario. For instance, displays twisted relative to each other, *i.e.*, connecting a left/right edge with a top/bottom edge, can be penalized when used for cross-display navigation. If possible, users may prefer traveling via an intermediate display. Also, penalties are applied if normal vectors of two adjacent displays are facing away from each other.

Resulting edge connections between adjacent display pairs are managed in a graph data structure. Connected edges of adjacent display pairs represent point-to-point mapping areas between two displays. To find the shortest path between two arbitrary displays in the environment, the graph data structure containing the individual displays needs to be consulted. We use the A* search algorithm [100] to find the path with the lowest cost through the graph of displays, given a start and a target display and, if available, a specific pixel on the start and target display. The cost function of the algorithm for the path through a node n is composed of two parts [100]:

$$f(n) = g(n) + h(n),$$

where $g(n)$ is the cost for the optimal path from the start node to the current node n and $h(n)$ is an estimate of the cost from n to the target node. For a display n, $g(n)$ is defined by:

$$g(n) = g(n-1) + d_n,$$

where $g(n-1)$ is the cost of the previous display (or 0, if n is the start display), and d_n is the distance from display n to the previous display. The cost $h(n)$ is estimated by the world distance between display n and the anticipated target display. The result of the

algorithm is a list of displays resulting in the shortest possible path with respect to the real world.

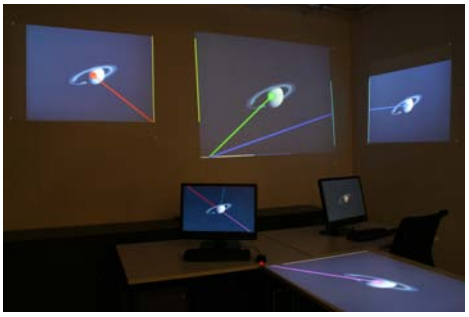


Figure 4.16: Visualization of the shortest paths from a start display (left monitor) to all possible target displays using pair-wise planar mappings and the A* search algorithm.

Figure 4.16 shows simple line connections from a single start display to all potential target displays. In this case, lines are rendered through the centers of the edge intervals of a display pair. This means that the length of the lines could be further optimized. To calculate the optimal position on the edge interval for a line connecting a display n and $n+1$, we project the start point of the line on display n onto the connected edge. Similarly, we project the end point of the line on display $n+1$ on the connected edge, and map it to the corresponding edge position on display n (Figure 4.17(b)). The position on the edge interval is determined by finding the midpoint between the two projection points and clamping it within the edge mapping interval, if necessary. Figure 4.17 illustrates the difference between the two approaches.

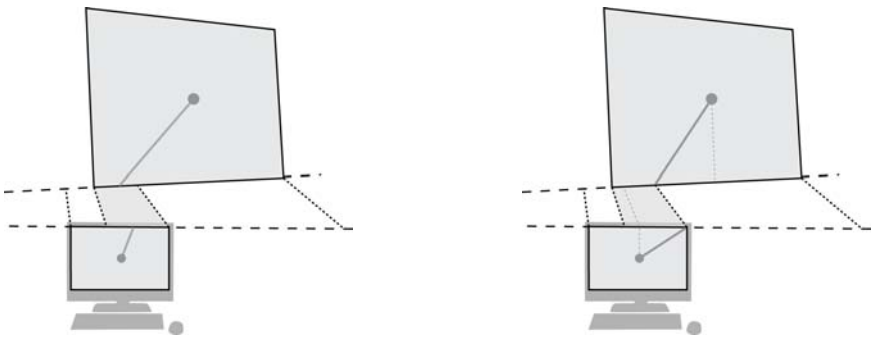


Figure 4.17: Line connections between display pairs: (a) through the edge connection midpoint or (b) the shortest possible path.

Pair-wise planar mappings can be employed in very complex environments, as the resulting coordinate systems are very simple and work equally well from all locations within the environment. They do not introduce any perspective distortions within the displays.

However, the possibilities of environment-aware cross-display visualizations are limited to very simple path visualizations, as shown in Figure 4.16. Using this mapping, it is not possible to create a seamless viewing space across more than two displays, as no unified 2D representation containing more than two displays is available. Another disadvantage is that pairwise planar mapping sometimes simply fail. For instance, if the right projected display was the only wall display in Figure 4.14, no connection with the monitors would be established with the above described algorithm. To circumvent this problem, the algorithm should detect display pairs that do not have a direct or indirect connection, and should add artificial connections between these displays. However, this features has not been implemented.

As extensions to the concept, pair-wise planar mappings could additionally incorporate the display-less space between the connected edges (*cf.*, Figure 4.13(c)). This representation would correspond to *seam-aware* visualizations [146], *OneSpace* multi-monitor imagery [200], or a *MouseEther* navigation coordinate system [18]. Mind that in their original implementation, these techniques have only been considered for co-planar display configurations. To create more meaningful edge intervals for a human observer, connected edge intervals could be derived from a 2D perspective mapping, as described in the next section.

4.4.2 Perspective Mappings

This mapping creates a perspective representation of the environment, as perceived from the (estimated) user location. A simple example is shown in Figure 4.13(d). In contrast to pair-wise planar mappings, this representation differs for each user or location in the environment, respectively.

From the known user location and viewing angle in the environment, a projection plane is constructed. Each displays' reconstructed polygon corners are mapped onto this projection plane. The screen coordinates of the individual polygons and their perspectively mapped representations are related by 2D homographies.

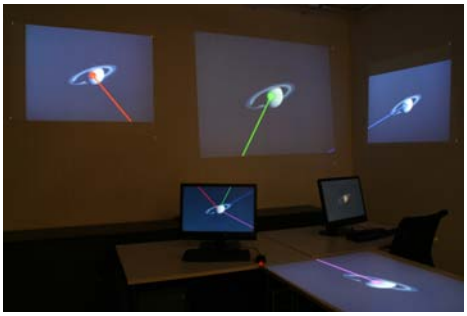


Figure 4.18: Visualization of the shortest paths from a start display (left monitor) to all possible target displays using a perspective mapping. Mind that the perspective plane corresponds to the camera position.

Figure 4.18 shows the same line visualizations as in Figure 4.16, but using a perspective mapping instead. To find the individual line segments for each display, the start and end point on the respective displays are first mapped onto the perspective representation. There, all intersections with display edges are detected and converted to the respective screen coordinate systems.

The perspective mapping leads to an intuitive 2D mapping of the environment, when viewed from the user's perspective. However, in a collaborative setting, a perspective viewing space may lead to confusion of observers in other locations. Also, if the user changes location or turns the head, the perspective mapping needs to be updated. Detecting these changes requires special sensing equipment, like head trackers, which is not only expensive, but also intrusive for the users. Although in some situations (*e.g.*, a static office setting) a static perspective mapping might be sufficient, head trackers are indispensable if some displays are located behind the user and can only be accessed if the user turns around.

In contrast to pair-wise planar mappings, visualizations can be rendered across an arbitrary number of displays, as a single unified 2D space is created. As an example, the *E-Conic* system [169, 207] used a similar perspective mapping to render application windows in an MDE.

4.4.3 Other Mappings

There are, of course, other possibilities to map an MDE onto a planar representation. The considerations below are purely conceptual and have not been integrated into the Deskotheque environment.

A distortion-free, but unified 2D mapping could be achieved by using an approach similar to the patch parameterizer introduced for mapping of non-developable surfaces to a 2D texture space (Section 4.1.4), resulting in a "fold-out" view of the environment: Pair-wise adjacent displays are determined similar as in the pair-wise planar mapping technique, but additionally the distance between their closest edges is encoded in the planar representation (as in Figure 4.13(c)). If the environment is non-developable, vertically aligned displays (connecting left / right edges) are preserved, while "twisted" connections along a lateral and top / bottom edge (usually involving a tabletop display) are potentially sliced. Figure 4.19 shows a conceptual sketch of this approach. As a unified planar mapping of the environment is available, cross-display visualizations are possible. This representation can be compared with the fold-out view of the *ARIS* window relocator [33]. However, in their implementation, the mapping had to be constructed manually [36].

One might also consider a perspective mapping onto a non-planar geometry, such as a sphere or a cylinder. Examples are the cylindrical parameterization for the surround display by Pirchheim *et al.* [180] or the spherical navigation frame of the *Perspective Cursor* [170, 207].

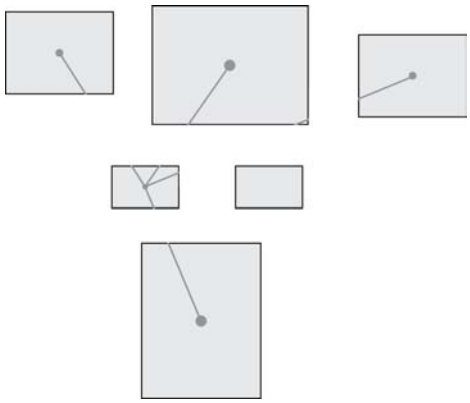


Figure 4.19: Conceptual sketch of a fold-out mapping of the environment in Figure 4.14, with cross-display lines.

4.5 Discussion

The automatic display calibration process allows for a quick and easy creation of a multi-planar model of the environment. Each displays' pixel can be uniquely mapped to its corresponding location in the 3D model of the environment, as well as any other 2D model of the environment, which has been derived from the initial reconstruction. The calibration process extends the approach for reconstruction of multi-planar displays presented by Quirk *et al.* [187] to more flexible multi-planar display form factors. The model created with the calibration procedure proposed by Quirk *et al.* was limited to displays with vertical display edges.

Our spatial model approach furthermore supports the combination of monitors and irregular, tiled projected displays into a common interaction space. This is especially useful in office environments with limited wall and table spaces, where providing perfectly aligned projections is hampered by architectural constraints. To our knowledge, Deskotheque has been the first MDE to combine monitors and irregular projections in a common, spatially aware interaction space. Wallace *et al.* [254] combined multiple single-monitor workstations and a tiled projected display in a control room setting, but their environment does not require any spatial awareness features: Input redirection and information transfer is fixed between the top edge of each monitor and the bottom edge of the shared projected display.

Our calibration process and description of the environment as polygonal model also suffers from a few limitations. First, we do not synchronize the cameras used for calibration with our display devices. This limitation is based on the design requirement for commodity hardware and the support for flexible hardware configurations. Thus, the calibration process is time-consuming and can only be conducted offline. Imperceptible structured light approaches cannot be supported in such a configuration. As a result, changes to the display environment cannot be detected at runtime and require the users to re-run the

calibration process manually. However, updating the model based on real-time tracking of display devices using displayed natural features, as proposed by Johnson and Fuchs [133], could be a useful future extension to provide a hybrid offline-online calibration procedure.

Second, due to similar considerations about hardware cost and flexibility, we do not support tracking of users and devices in the environment. According to our own informal and observational experiences, we concluded that real-time tracking of environment resources is rarely required in office room scenarios. However, as handheld display devices are becoming more and more ubiquitous and light-weight (*e.g.*, tablet PCs), continuous 6-DOF tracking of certain display devices will be an important future issue. Natural feature tracking of displayed content, as mentioned above, could be a solution, but may be too unreliable. For reliable 6-DOF tracking with sufficient accuracy, attaching tracking targets to mobile devices, such as fiducial markers or infrared targets, will be inevitable.

Third, in its current implementation, the calibration process relies on calibrated multi-view vision to create a reconstruction of the environment. With this configuration, most environments can be reconstructed. However, in many situations, a 3D model could be created much easier. For instance, when limited to a single planar display surface, as shown in Figure 4.9(a), a single camera mounted perpendicular to the display surface would be sufficient for an accurate reconstruction. Similarly, if monitors with known aspect ratio and display diagonal are available in the environment, the 3D coordinates of the displayed structured light points on the monitors can be calculated. From the point correspondences of the captured structured light feature points in camera coordinates and the associated 3D coordinates, the internal camera parameters can be estimated. If the radial and tangential lens distortion is low, this estimate may be sufficient for a reconstruction. Especially if there are no tiled displays, the reconstruction for creating a seamless interaction space can be fairly coarse.

Fourth, the physical display arrangement of the Deskotheque environment is limited to multi-planar displays. Projection on arbitrary, *e.g.*, curved, surfaces is not supported and will lead to unpredictable geometric compensation. In addition, Deskotheque assumes that displays are discontinuous and that each individual display can be mapped to a 2D representation. Closed, seamless surround projections are not considered in office spaces, where architectural elements like doors, windows, or posters lead to a natural segmentation of the available display regions.

Part III

Multi-Display Navigation

Chapter 5

Cross-Display Navigation Techniques

Cross-display navigation is a fundamental interaction technique to facilitate information access in MDEs using indirect pointing devices, like a conventional computer mouse. Considering the mouse as primary input device in an MDE might seem "old-fashioned" for some interaction designers, but has advantages for operating a personal device-centric MDE, such as illustrated in Figure 5.1: Mouse and keyboard are the standard input devices for personal computers – the ones considered to the central interaction hub of the user – and can be similarly employed for operation on distant content on projected displays. Users can access very close, and detailed content, as well as very distant items, without physically moving to a remote display [33] or changing input device. On shared displays, interference of multi-user input is kept to a minimum, as users do not occlude displayed content when interacting on public display space – in contrast to direct input methods. Finally, today's legacy application operation is best supported by mouse and keyboard, and input hardware has a very low cost, compared, for instance, to large touch-sensitive surfaces.

Figure 5.1: Personal device-centric MDE operated by a mouse and keyboard pair.

However, providing a seamless navigation space across all discontinuous displays in the environment poses interesting challenges. First, as there is a large amount of pixels available in the environment, we are facing similar problems as continuous large displays: users have problems keeping track of their cursors and moving the cursor across the environment introduces considerable physical effort [68, 200]. To overcome these issues, visual navigation aids for large displays and improvements of targeting performance have been addressed (see Section 2.2.2).

Second, the environment is composed of multiple heterogeneous displays of different resolution, size, orientation, and with considerable display-less gaps between the individual displays. Previous research has shown that seamless mouse pointer navigation is negatively influenced by a number of display factors, such as depth offsets between displays [237], adjacent displays at relative angles higher than 45° [233], physical distance and size-resolution mismatches [18], as well as non-optimal seating arrangements [257]. In these evaluations, standard mouse behavior was used for navigating across heterogeneous displays: multiple output devices are mapped into a common planar coordinate system, not considering resolution, distance, or orientation (*cf.*, Figure 4.13(a)).

Cross-display navigation techniques therefore need to keep the user's mental effort low by additionally addressing the following issues: Providing intuitive *transitions*, where the mouse pointer is relocated from one display to another, and compensating for heterogeneous mouse movement behavior caused by changing display factors.

5.1 Design Space

From reviewing the related work on cross-display navigation (Section 2.2.3), it is evident that mouse pointer navigation in MDEs covers a large design space. We separate the design of cross-display navigation techniques into two major areas: The fundamental requirement is the creation of a multi-display coordinate system the navigation technique can refer to. This corresponds to the *display configuration* established by Nacenta *et al.* [166], and exemplary approaches are described in Section 4.4. The spatial model can be created in an automated calibration step, modeled manually by the user, or interactively created by the user.

Based on this coordinate system, the navigation technique can apply different navigation parameters. We introduce four categories describing differences in navigation techniques, which directly informed the system design of the multi-display navigation infrastructure presented in Section 5.2:

Trigger. A cross-display navigation technique somehow needs to trigger a transition of the pointing device from one display to another. On the one hand, this trigger might be *implicit* as the user moves her pointer across a display edge (*e.g.*, *MouseEther* [18], *Perspective Cursor* [170]), or a specific interval on an edge (*e.g.*, *PointRight* [129], *Swordfish* [96]), and no additional activity is required. On the other hand, an *explicit*

trigger requires the user to undertake some dedicated action to transfer the pointer to a remote display. Examples are graphical representations of the environment (*e.g.*, *ARIS* [33]), textual lists of display names, or button presses on mouse or keyboard (*e.g.*, M^3 [28, 29]). For some explicit triggers, no spatial representation of the environment is required.

Cross-display movement. When transferring the pointer to a remote display, the display-less space between source and target display needs to be bridged. The pointer may either be *warp*ed across the gap, or may *continuous*ly move in display-less space, *i.e.*, within the *ether* [18, 170]. Continuous movement in display-less space is usually supported by an off-screen visualization technique, to indicate the approximate location of the pointer. To decrease the amount of time spent in display-less space, movement speed can be accelerated. To properly model movement in display-less space, it needs to be encoded in the spatial representation of the environment. Techniques facilitating an explicit trigger directly transfer the pointer to a remote display and therefore do not create the illusion of a seamless navigation space. Thus, warping the pointer to the remote display is the standard approach for explicitly triggered techniques.

Outcome. When arriving at the remote display, the navigation technique needs to set the pointer to a reasonable initial position. Techniques using an implicit trigger usually set the mouse to a *display edge* – either by evaluating point-to-point mappings of pair-wise planar mappings [96, 129], by extrapolating a movement direction and intersecting the resulting ray with a potential target display edge, or by simply setting the pointer to the first known position after continuous movement in display-less space [18, 170] (Figure 5.2). For techniques with explicit trigger, a straight-forward approach is to set the pointer to the *center* of the remote display or some other pre-defined location. Benko and Feiner [28, 29] presented some possible outcome strategies for techniques with explicit triggers. Others let the user explicitly choose the target location in a *GUI* preview of the display [33].

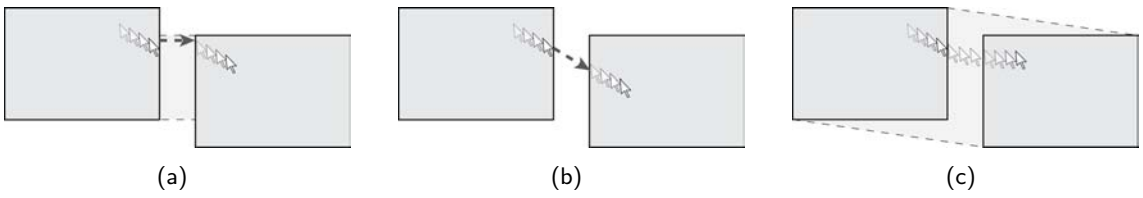


Figure 5.2: Potential outcome positions on the remote display's edge after an implicitly triggered transition: (a) point-to-point mapping along virtually connected edges, (b) extrapolating the movement direction, or (c) entering the display after continuous movement in the display-less "ether".

C/D gain. The final category addresses movement within a display, as opposed to cross-display movements. The C/D gain adjustment deals with the question how to compensate for non-homogeneous movement across multiple displays. The *standard* C/D gain applies the operating system's default behavior, which does not take physical display size, distance to the user, or viewing angles into account. Potential ways to adjust the C/D gain in MDEs is to compensate for *resolution* differences of displays [18] or to fully compensate for *perspective* mappings onto the user's viewing plane [170], which requires a perspective representation of the environment. To compensate for inhomogeneities, each relative incoming mouse movement event needs to be evaluated on the spatial model and corrected, if necessary. Mind that the C/D gain adjustments considered here only affect the interaction space. Similar considerations may be conducted for the viewing space (*cf.*, *E-Conic* [169]).

Based on our multi-display coordinate systems (Section 4.4) and the above established criteria, we developed a generalized technical infrastructure which covers most of the design possibilities in a unified way. This infrastructure is introduced in the next section, followed by some exemplary navigation techniques, implemented using this infrastructure.

5.2 Mouse Pointer Navigation Infrastructure

To support a wide range of cross-display navigation techniques, we implemented a navigation infrastructure operating on three different layers (Figure 5.3):

- Optional external applications can be used to explicitly trigger input redirection commands. Applications facilitate Deskotheque's network interface to report input redirection triggers. As examples, we provide an interactive world-in-miniature control of the environment (Section 5.3.3) and a simple pointer warping technique (Section 5.3.4), which allows the user to relocate the pointer to a dedicated target display by pressing a keyboard shortcut. In addition, external applications can also receive navigation events, for instance that an input redirection has occurred for a specific pointer. For instance, in the Deskotheque environment, this information is incorporated by the window manager to visualize the pointer outcome position on the target display as an animated dot.

- The navigation framework receives motion events from a single pointing device, performs implicit transitions to remote displays, receives explicit transition requests from external applications, and applies C/D gain adjustments on demand.

- The base layer grabs input events from connected pointing devices. In our Deskotheque environment, we use a middleware application taking care of input redirection and multi-pointer coordination, as described in Section 3.2.2. For each detected mouse pointer, one instance of the navigation framework is assigned. In

this way, different pointers and users, respectively, can use distinct spatial representations of the environment or navigation parameters for their preferred cross-display navigation behavior.

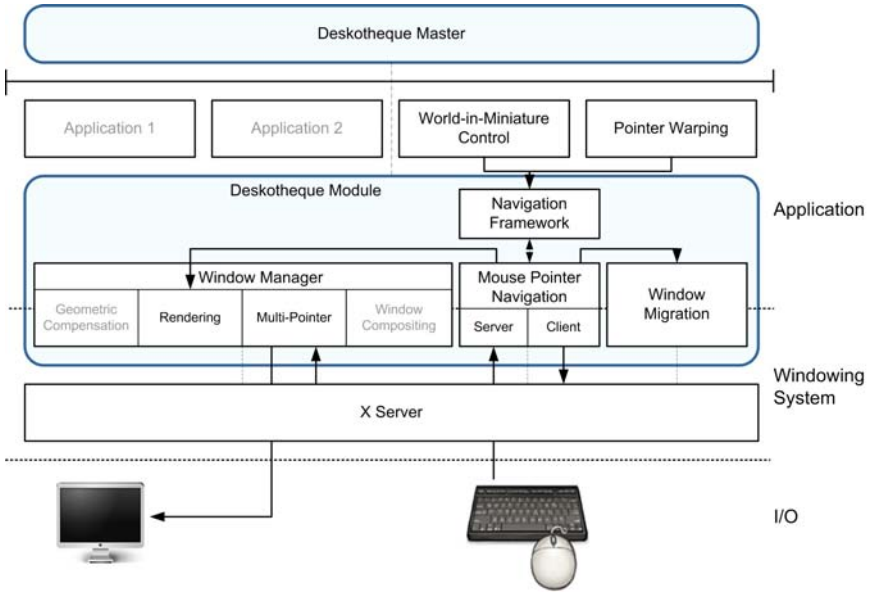


Figure 5.3: Deskotheque navigation infrastructure for a single pointer: The base input redirection component grabs input events from connected pointing devices and consults the navigation framework for each incoming input event. External applications can issue explicit redirection triggers. The window manager is responsible to render color-coded pointers and navigation aids, triggered by the input redirection component. If required, the window migration component is informed if an input redirection was triggered for a pointer dragging a window.

The remainder of this section will deal with the middle layer, the navigation framework. The primary purpose of the navigation framework is to compensate for inhomogeneities in the navigation space (*i.e.*, to apply C/D gain adjustments) and to perform transitions to remote displays, which were implicitly triggered. It also provides an interface for external applications to perform explicit triggers. The navigation framework can then choose an appropriate target display and an outcome position, if not given. Explicitly triggered transitions can co-exist with seamless mouse pointer navigation, triggering transitions implicitly by moving the pointer across display boundaries.

5.2.1 Navigation Framework Configuration

The navigation framework is configured with a spatial description of the environment, as well as parameters of the desired navigation behavior. The spatial model describes each available display in screen coordinates, in combination with potential edge connections to adjacent displays, and a representation of this display in a unified 2D representation, *e.g.*, a perspective view or a fold-out view of the environment (Section 4.4). In our system, the

connected edge intervals and the 2D map of the environment are automatically retrieved from a 3D model of the environment, created in an automated offline calibration process (Chapter 4).

The desired navigation parameters were derived from the categorization of the design space described in Section 5.1. For implicitly triggered navigation techniques, the user has to specify an *exit* trigger. This might be if the mouse touches a connected edge interval or if the user touches a display edge at any position.

In addition, a *transition* type has to be specified. The transition determines the target display and the respective outcome position. Currently, we support three types of transitions: Connector transitions set the pointer to the complimentary interval on the remote display, if the exit was triggered by a connected edge interval configuration. Map transitions use the unified 2D map of the environment to determine target display and outcome position – either by finding the remote display location closest to the current mouse pointer location or by extrapolating the current pointer movement direction and intersecting the resulting ray with the displays on the 2D map. Finally, sequential transitions hold a pre-defined sequence of displays and set the mouse pointer to the center of the next display in the sequence. A transition type also has to be specified for externally triggered transitions, in case the external application does not specify the target display.

Finally, the *outcome* parameter overwrites the outcome position specified by the transition type, and is usually desired for externally triggered transitions, such as pointer relocations triggered by shortcuts. We support some simple strategies, as proposed by Benko and Feiner [28, 29]: centering the mouse pointer on the remote display, setting it to the same relative location, and setting it to the same location as previously on this display.

In addition, the user can specify a chain of *C/D gain* adjustments. If no adjustment is applied, the mouse will behave as usual when moved inside a display. In our implementation, the user can choose to compensate the C/D gain so movements are homogeneous with respect to the underlying map representation (*i.e.*, a non-linear distortion of the interaction space, if displays in the map are perspectively distorted).

This approach offers a variety of configuration possibilities. For instance, setting connected edges and connector transitions for implicitly triggered transitions, we can easily model multi-monitor "stitching", as proposed by *PointRight* [129] or *Swordfish* [96]. The combination of edge exits and map-based transitions, together with map-based C/D gain adjustments, creates similar effects as *MouseEther* [18] or *Perspective Cursor* [170]. Mind, however, that our navigation framework always warps the mouse pointer across display-less space. Movement in the display-less "ether" is not yet supported. In addition, we can also create combinations of these techniques. For instance, we can restrict the exit regions of a display to pre-defined connected edge intervals. If the pointer touches such a region, a map-based transition extrapolates its movement direction and relocates the pointer to the respective remote display edge intersection.

For explicitly triggered transitions, very simple standard pointer warping approaches

can be easily constructed, for instance by combining sequence transitions with center outcome (Figure 5.4(a)). In this case, the pointer will be re-set to the next display's center in the linear sequence by pressing a dedicated trigger. Also, more complex settings can be constructed. For instance, the user may trigger a pointer warp with a single button press and relocate the pointer to the closest edge of a display in its current movement direction (map-based transition), as illustrated in Figure 5.4(b).

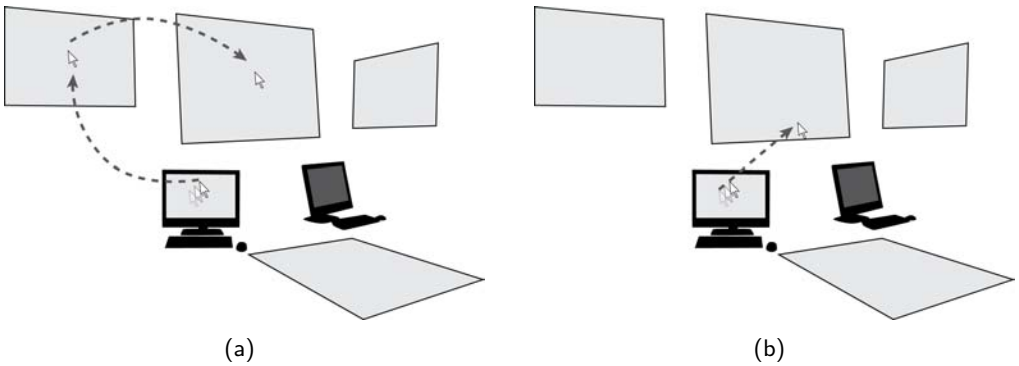


Figure 5.4: Potential pointer warping strategies with a single button press: (a) sequential switching through displays and centered outcome position or (b) extrapolating the current mouse pointer movement direction after explicitly triggering input redirection.

5.2.2 Runtime Operations

At runtime, the base layer constantly delivers movement events of a single input device to the navigation framework. In our environment, the Synergy server grabs input events from multiple input devices. For each input event, the navigation framework associated with the particular pointing device is informed. If the navigation framework somehow modified the incoming pointer location (*e.g.*, by applying C/D gain adjustments) or requests a transition to a remote display and host, respectively, the base layer is responsible to apply these operations to the respective pointing device.

After receiving the latest motion event from the base layer, the navigation framework first applies the C/D gain adjustment chain. Then, it checks whether the requested display exit criteria is fulfilled, *i.e.*, whether the mouse pointer touches an edge or a virtually connected edge interval of the display. If so, the user-specified transition method determines the target display and outcome position on the display. If specified by the user, the outcome position is overwritten, *e.g.*, to always re-set the mouse pointer to the center of the display. Figure 5.5(a) illustrates this procedure.

When receiving external trigger requests from external applications, there are three potential cases: First, the application already determined the respective target display and the exact location on the target display. This is, for instance, the case when picking the desired target location in a miniature view of the environment. In this case, the navigation

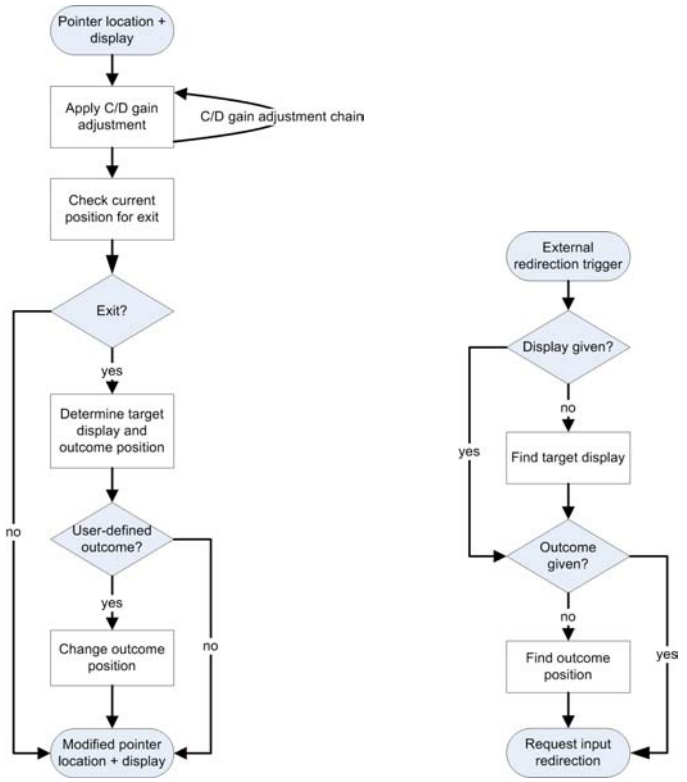


Figure 5.5: Flow diagrams of runtime operations: (a) applying C/D gain adjustments and checking for implicit input redirection triggers when receiving the current pointer location from the base layer and (b) forwarding an input redirection request to the base layer after receiving an external trigger.

framework just updates its internal pointer history and reports the change to the base layer. Second, the application reports the target display, but does not report a specific location. This is the case if the user selected the target display from a simple textual list or by pressing a dedicated shortcut. The outcome position is then determined using the outcome parameter set for explicitly triggered transitions. For instance, if the user chooses a relative placement strategy [28], the mouse pointer will be set to the specified display to the same relative position as on the origin display. Third, the application neither reports the target display, nor any location. This is most likely if the user just presses a single button to cycle the pointer sequentially through all displays. The navigation framework then consults the transition method specified for external triggers and afterwards sets an appropriate outcome position. Figure 5.5(b) shows the operation flow when receiving an external redirection trigger.

5.3 Exemplary Navigation Techniques

Based on our design criteria for cross-display navigation and the established mouse pointer navigation infrastructure, we built four different cross-display navigation techniques for heterogeneous MDEs. Table 5.1 illustrates how these techniques fit into the design space.

	Path Navigation	**Free Navigation**	**WIM Control**	**Pointer Warping**
Spatial model	pair-wise planar	perspective	3D	none
Trigger	implicit (edge interval)	implicit (edge)	explicit (GUI)	explicit (key-press)
Outcome	point-to-point edge-mapping	ray-edge intersection	selected in GUI	display center
C/D gain	standard	perspective	standard	standard

Table 5.1: Our developed techniques differ in which spatial model they use, how a transition is triggered, the outcome position on the target display, and the C/D gain adjustment. All techniques warp the mouse pointer across display-less space.

5.3.1 Path Navigation

Path navigation establishes mouse pointer transitions along virtually connected edge intervals, derived from pair-wise planar mappings of the environment (Section 4.4.1). The navigation framework triggers an implicit exit if the mouse pointer touches a connected edge interval and transfers the pointer to the complimentary edge position on the remote display, as illustrated in Figure 5.6. Movement within displays operates with standard C/D gain, but could also be adjusted to compensate for perspective effects or resolution differences, if the user desires. To support the users in path finding, we visualize connected edge intervals by color-coded lines.

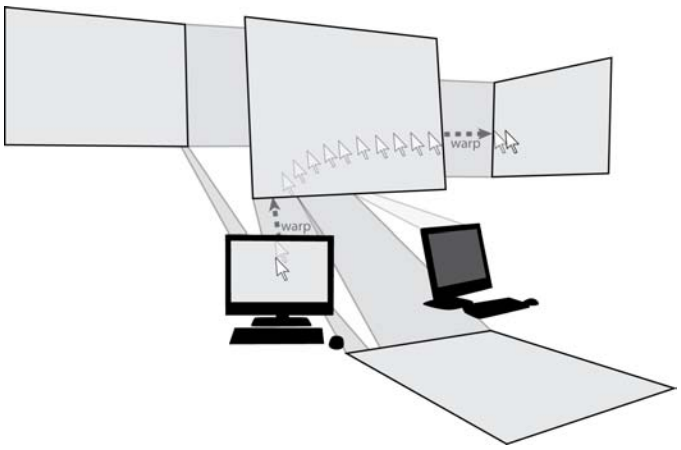


Figure 5.6: Path navigation warps the mouse pointer across virtually connected edge intervals in pair-wise planar mappings.

Path navigation can be compared to conventional "stitching", as provided by most operating systems to handle multi-monitor settings, or input redirection tools like *Synergy* [212]. However, unless configured manually, these tools do not consider the spatial arrangement of displays towards each other or resolution differences in display-connecting paths. Path navigation allows for more flexible display arrangements, like multiple connections along a single edge (*cf.*, the bottom edge of the center wall display in Figure 5.6) or twisted connected edges (*cf.*, the connection between table and central wall display in Figure 5.6).

5.3.2 Free Navigation

Free navigation works similar to *MouseEther* [18] or *Perspective Cursor* [170]. Each incoming relative mouse movement event is added to the last known pointer position on a perspective representation of the environment, as seen from the user operating the mouse (perspective C/D gain).

However, there are two major differences to MouseEther and Perspective Cursor: First, free navigation does not consider the display-less space within the displays but rather warps the pointer across the gap when transitioning to a remote display. Although an evaluation of Perspective Cursor [170] showed that perspective navigation within MDEs is superior to conventional stitching, a follow-up experiment [168] suggested that warping the pointer across a large gap between displays performs better than navigation in the mouse "ether". As MDEs in offices or meeting rooms often include displays with considerable physical gaps in between, we therefore refrained from navigation in display-less space. Instead, each time the pointer touches a display edge, the current movement direction is extrapolated to a ray and intersected with the other displays' edges in the perspective map, as shown in Figure 5.7. If the ray intersects other displays' edges, the intersection point closest to the current

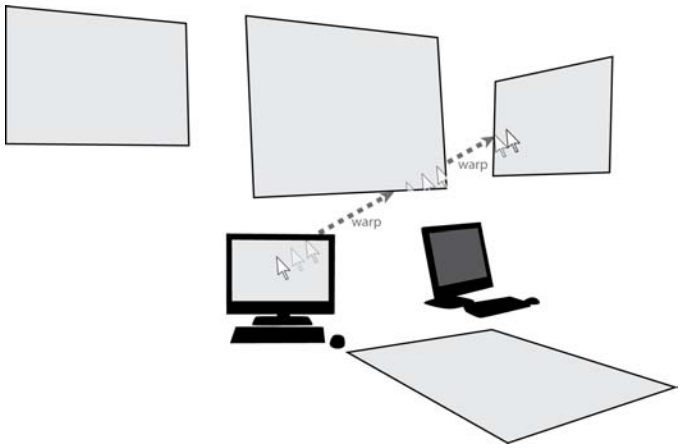


Figure 5.7: Free navigation operates with a perspective C/D gain and warps the mouse pointer across display gaps by extrapolating its movement direction on a perspective map of the environment.

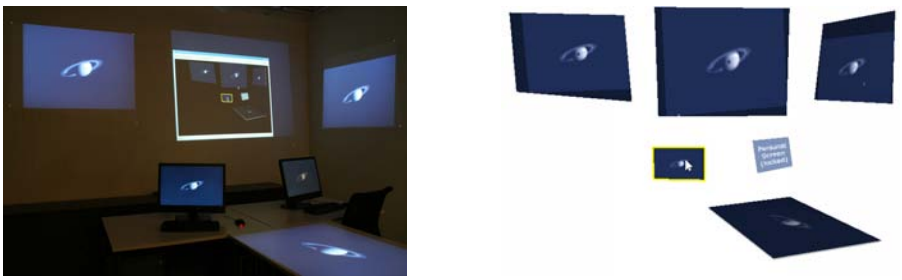
position is converted to the target display's device coordinates and input redirection is triggered. If no intersection was found, the motion event is discarded, and the mouse pointer remains on the originating display.

As a second difference to Perspective Cursor, free navigation uses a static description of the environment, created in an offline calibration step with an estimated user location (Section 4.4.2). This approach only allows operating the entire environment, if all displays are within the user's field of view. Otherwise, some displays may be impossible to reach or the mapping needs to be changed as the user moves her head.

5.3.3 World-in-Miniature

A world-in-miniature (WIM) approach is a well-known concept for accessing and modifying content in virtual environments [227], but has also been used for input and content redirection in MDEs [33, 265] or large displays [234]. A WIM represents a down-scaled view of the environment and thereby aids discovery, access, and manipulation of remote content.

Our WIM control is an external 3D application based on *Coin3D*[*] to trigger input redirection interactively on the 3D environmental model created in the offline calibration process. Display models are textured with live desktop content, which is retrieved and streamed by the window manager. The virtual camera location of the WIM control is initiated from the estimated perspective of the user invoking the WIM control and can be interactively modified. The WIM control is shown in a small window, which is invoked by a shortcut and appears at the current mouse pointer location. Figure 5.8 shows the WIM control window.



(a) (b)

Figure 5.8: (a) The world-in-miniature control window appears at the current mouse pointer location and (b) allows the user to interactively select the pointer target location in the environment (screenshot close-up).

To redirect the mouse pointer input, the user clicks the desired target location within the miniature view. As a result, the navigation framework is informed about the anticipated target display and location and initiates an input redirection to the associated

[*] `http://www.coin3d.org/`

location in the physical environment.

5.3.4 Pointer Warping

Pointer warping triggers mouse pointer transitions by pressing a dedicated keyboard shortcut. As sequential switching by pressing a single button becomes cumbersome with an increasing number of displays, we allow the user to explicitly redirect the pointer to a dedicated display. Each display in the environment is issued a unique ID, which can be visualized on demand, as illustrated in Figure 5.9. A transition to a desired display is achieved by pressing a modifier key in combination with the display ID. The outcome position is set to the center of the display.

Figure 5.9: On-demand visualization of the display IDs used for pointer warping.

In contrast to the other exemplary navigation techniques, pointer warping does not need to take the spatial display arrangement into account. The display is selected explicitly based on a unique identifier and the outcome position is not influenced by the origin location within the environment.

5.4 Cross-Display Information Sharing

Cross-display navigation techniques are not only useful to transfer input control to a remote display, but can also be used to transfer pieces of information (*e.g.*, application windows) to a remote display. Navigation techniques using an implicit trigger can be utilized for drag-and-drop of information items across display boundaries, just like windows can be dragged across monitor bezels in conventional multi-monitor settings. In a distributed setting, information sharing or application window migration has to be triggered whenever the mouse pointer, which is currently dragging a window, is redirected to a remote host (see Section 3.2 for a technical description). In a single-machine setting, the dragged window simply has to follow the mouse pointer, even if it is warped by the navigation framework.

Window migration can also be triggered explicitly. For instance, the world-in-miniature control can be facilitated to additionally support drag and drop of application windows

across display boundaries within the miniature view, as realized in a single-machine prototype for a compositing window manager [24] (*cf.*, Figure 5.10). As another explicit window migration trigger, we embedded a list of potential target display names into the context menu of the window title bars in an earlier version of Deskotheque [181].

(a) (b) (c)

Figure 5.10: Window migration using the WIM control [24]: (a) the user can pick up a window at any location, (b) drag it to a remote display within the miniature view, and (c) drop it on a remote display. The window is resized to fit the display size. The window migration operation will be reflected in the "real" environment.

5.5 Discussion

The presented mouse pointer navigation infrastructure is a modular approach to cross-display navigation and therefore provides for flexible configurations. The infrastructure separates low-level input event handling, like multi-pointer support and input redirection in a distributed system, from high-level navigation control, as well as external applications providing special features. The environment configuration is provided by an external source, thus can be created by hand in a simple configuration file (*e.g.*, as in *PointRight* [129] or *Synergy* [212]), in a graphical editor (as proposed by Biehl and Bailey [36]), interactively by the user (as in *Swordfish* [96]), or from an automated calibration process, as described in Chapter 4.

In its current implementation, the navigation framework relies on a spatial model of the environment, which has been created offline and is not changed at runtime. Modifications to the model at runtime are required for instance when using a perspective mapping in combination with a head-tracked user, if tracked mobile devices are available in the environment, or static displays are added or removed at runtime. Conceptually, supporting a dynamic spatial model is possible, but has not been required for our exemplary navigation techniques.

The proposed design space currently only deals with indirect pointing devices. Although the usage of indirect pointing devices as primary input control in a heterogeneous MDE has already been motivated at the beginning of this chapter, the increasing amount of self-contained devices with touch-sensitive input affords new possibilities in such an environment. In the future, hybrid approaches may be possible, where a tablet PC or

another handheld device is facilitated to explicitly trigger a transition, while on-display navigation is accomplished with the mouse, as usual.

Chapter 6

Evaluations of Multi-Display Navigation

A key aspect of the cross-display navigation techniques presented in the previous chapter (Section 5.3) is that they all facilitate a different spatial model of the environment. In previous work, different aspects of spatial awareness in cross-display navigation techniques have been evaluated. For instance, Baudisch *et al.* [18] showed that compensating for the visual-device space mismatch in dual-monitor setups (discontinuities caused by monitor bezels and size-resolution mismatches) increased targeting performance. Similarly, Nacenta *et al.* [170] demonstrated a benefit of a perspective navigation space in a heterogeneous MDE. Others have evaluated influencing display factors on navigation performance, such as depth offsets [237], angles between displays [233], and seating arrangements [257] – albeit without adapting the mouse pointer navigation to the spatial display factors. What is missing is a thorough understanding of how different display factors influence the cross-display navigation techniques with their diverse spatial models.

We started by rather informally comparing four navigation techniques introduced in the previous chapter (Section 5.3) on a heterogeneous MDE with different transitions across varying display factors (Section 6.1). Results of this experiment serve as foundation for further, more focused investigations. In particular, we explored the influence of physical display-less space on pointer warping (Section 6.2) and investigated whether making pointer warping spatially aware can enhance pointing performance (Section 6.3).

6.1 Comparison of Cross-Display Navigation Techniques

We compared performance, user satisfaction, and usage frequencies of four cross-display navigation techniques in a heterogeneous multi-display environment to receive quantitative measurements for comparison of the techniques, but also to receive qualitative feedback and exploratory evidence of navigation parameters and display factors influencing cross-display navigation. The four evaluated cross-display navigation techniques are presented

in Section 5.3: path navigation (*path*), free navigation (*free*), world-in-miniature (*WIM*), and pointer warping (*warp*).

Mind that the four navigation techniques not only differ in how they encode spatial awareness of the environment, but also in other navigation parameters (refer to Table 5.1 in Section 5.3).

6.1.1 Participants

Twenty users (11 male, 9 female, aged 23 to 48) participated in the study. Twelve participants were using dual-monitor setups on a regular basis, one participant was working with a three-monitor setup, and seven were using a single monitor. Half of the users ranked their computer experience five on a 5-Point Likert scale. Six participants assessed their knowledge level below four.

6.1.2 Setup

The experiment was conducted on a static setup consisting of six displays (Figure 6.1). Two single-monitor workstations were placed on a table, where one monitor was private and not accessible for the participant. Additionally, there was a tabletop projection separating the two private workspaces and three wall projections. Each projected display had 1024x768 pixels, while the monitors had a higher number of pixels (1920x1200 pixels). Size varied from 24" for the monitors to approximately 85" for the large, intermediate projection wall. Each display was driven by a separate machine, connected via gigabit network.

The users were seated in front of the left workstation monitor, with mouse and key-

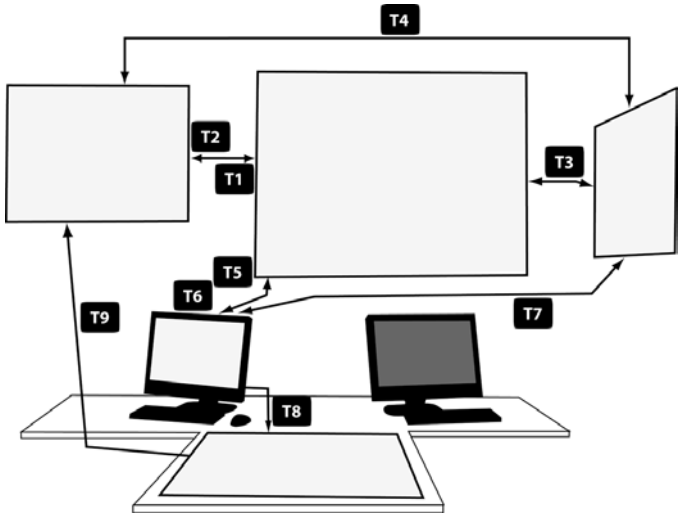


Figure 6.1: Experimental setup with the nine transitions (T1-T9) evaluated in the experiment. The user was sitting in front of the left monitor. The right monitor was inaccessible.

board placed in front of the monitor. There were no occlusions of displays from the user's point of view. Mind that the tabletop display is partially located outside the user's field of view, so the estimated user location employed for free navigation was corrected to include all displays within the user mapping. The desktop content of the tabletop display was oriented towards the table edge where the user was seated.

The setup and the evaluated transitions were chosen so that different display factors were covered. Table 6.1 illustrates how the individual transitions are associated with display factors: Almost all transitions connect displays of different size (and resolution). Most displays are not oriented at the same angle (co-planar), and some are located on different depth levels. Three transitions require crossings of (at least) one intermediate display with the path navigation technique, and in three transitions one display (the tabletop displays) lies partially outside the user's field of view. These transitions additionally differ from the others, as non-opposite edges (*e.g.*, bottom to left) are connected in the path navigation technique. Of course, this setup does not cover all potential display factors. For instance, (partially) occluded displays have not been included.

Transition	Size change	Orientation change	Depth change	Display crossing	Outside FOV	Non-opposite edge
T1	S>L					
T2	L>S					
T3	S<>L	1				
T4		1		1		
T5	S>L	1	C>D			
T6	L>S	1	D>C			
T7	S<>L	1	C<>D	1		
T8	S>L	2	C>B		T	B>L
T9	L>S	2	B>D	1	S	L>B

Table 6.1: Display factors influencing cross-display transitions T1-T9: whether the two displays differ in their size (from small = S to large = L, large to small, or both ways for bi-directional transitions), on how many axes the orientation of the two displays differ, whether the two displays have a varying distance to the user (depth change between close = C, distant = D, or behind the user = B), how many intermediate displays have to be crossed, whether one display (source = S or target = T) lies partially outside the user's field of view, and whether non-opposite edges are connected for path navigation (left = L, right = R, bottom = B, top = T).

6.1.3 Task

For each of the four navigation techniques, users were asked to accomplish a target selection task. Targets were small windows with a 225x178 pixels push-button appearing sequentially at different display locations. The push-buttons were textured with a centered cross-hair and alien space ships, which "invaded the desktop". Users were instructed to get rid of the aliens as quickly as possible and received scores for target selection time, as well as penalty scores for inaccurate selection of the cross-hair center. In other words, users were asked to select targets as quickly and centered as possible. Figure 6.2 shows

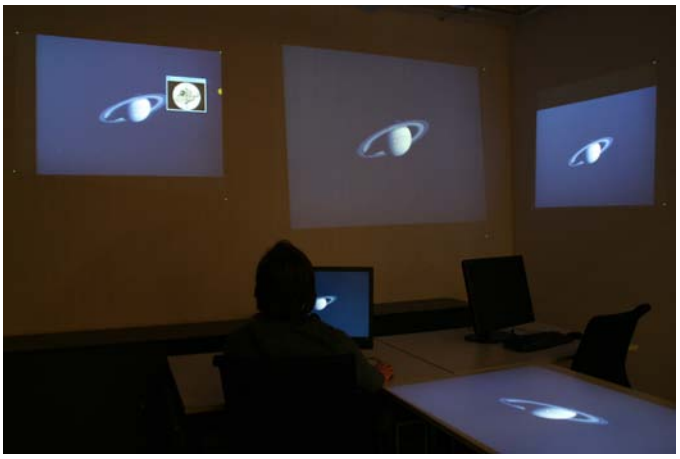


Figure 6.2: Alien ship target to be selected as quickly and centered as possible.

the experimental setup and an alien space ship target on one of the displays.

The system does not apply content size correction, thus physical target size varied across displays. According to the target appearance sequence, we separated each run into the nine cross-display transitions. In sum, twenty targets had to be selected per run, including some duplicate transitions. We measured task completion times between two successive target selections.

In retrospective, this task setup was not ideal as users did not get the chance to rest between successive target selections and some – but not all – transitions had to be crossed multiple times in order to get at least one measurement for each of the nine transitions T1 to T9. Dedicated start and target buttons with performance measures only between start and target button selection would have been more appropriate.

6.1.4 Design and Procedure

The study was conducted as single user experiment and employed a 4x9 within-subjects factorial design with the following factors:
navigation technique: path, free, WIM, warp and
transition: T1-T9.

As dependent variable, we measured the time between two target selections for each of the nine transitions. Additionally, we collected subjective questionnaire evaluations (seven-point Likert scales) and qualitative feedback.

Every run was preceded by a short training phase where users could get familiar with the navigation technique. The order of navigation techniques was counterbalanced to minimize learning effects. The sequence and location of targets was pre-defined in different sets, which were assigned to the four navigation techniques in a balanced fashion.

Users first had to finish twenty target selections for each navigation technique. After finishing the four techniques, users were asked to accomplish a combined navigation

condition. They could choose between path navigation and free navigation as implicitly triggered navigation technique and could invoke the WIM control and pointer warping at any time additionally.

We collected task completion times for every consecutive target selection, continuous mouse pointer logging data, and video-taped the session. At the end of the first four runs, users were asked to fill out a questionnaire evaluating the four navigation techniques. For the combined condition, we determined which navigation technique was used for the individual transitions. An informal, semi-structured interview was conducted at the end of the experiment.

6.1.5 Results

Measurements were collected for 20 participants in four navigation techniques with nine tested transitions, resulting in 360 data points in total. We only considered the first occurrence of each transition, *i.e.*, for duplicate transitions subsequent measurements were discarded. One user test was declared as outlier and not included in the performance results.

Main effect and interaction analysis were performed at $\alpha = .05$ and Bonferroni adjustments were applied for post-hoc comparisons.

Task Completion Times

A 4 (navigation technique) x 9 (transition) repeated measures ANOVA on target selection times showed main effects for navigation technique ($F_{3,54} = 33.801, p < .001$) and transition ($F_{8,114} = 18.439, p < .001$). There is also an interaction between navigation technique and transition ($F_{24,432} = 8.099, p < .001$).

Post-hoc comparisons revealed that WIM was the slowest technique (average completion time across all transitions $t_{wim} = 5.84s$), followed by free navigation ($t_{free} = 4.17s$), which was slower than both, path ($t_{path} = 3.67s$) and warp ($t_{warp} = 3.29s$). Figure 6.3 shows the task completion times for each transition.

Compared to the other navigation techniques, pointer warping had the best performance for navigation between the monitor and the projected wall displays, as well as for navigation from and to the tabletop display. Both implicitly triggered navigation techniques (path and free) were faster for navigation between the wall displays (T1-T4) than WIM and warp. There was no performance difference between path and free for most transitions between two adjacent wall displays (T1 and T3 – for T2 path was faster than free) and the two outermost wall displays (T4), where the center display had to be crossed additionally. However, performance of path and free differed when more complex transitions than traversing between wall displays had to be accomplished: Free was superior compared to path for navigation between monitor and and the right wall display (T7), as well as when navigating from the tabletop display to a wall display (T9). However, it suffered from severe performance fallbacks when navigating *to* the tabletop display and

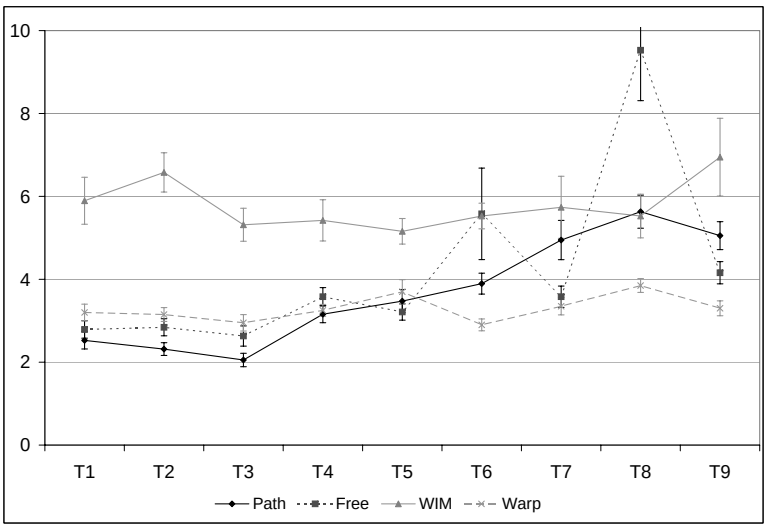


Figure 6.3: Task completion times in seconds of the four cross-display navigation techniques (average and standard error) for the transitions T1 to T9.

subsequently selecting a target there (T8). The fastest technique to access the tabletop was pointer warping. T8 was also the only transition, where the WIM control was not significantly slower than path and free. WIM and warp had almost uniform task completion times across all display crossings, thus did not seem to be strongly affected by changing display factors, as compared to path and free.

Usage Frequencies

In the last run – the combined condition – we gave the users the choice between path and free navigation and let them use WIM and warp additionally whenever pleased. When having the choice, eleven out of twenty users decided for path navigation, nine for free navigation as implicitly triggered technique.

For each transition, we logged whether it was performed with an implicit transition technique (path or free), pointer warping, the WIM control, or combinations (*e.g.*, pressing an incorrect shortcut for pointer warping and completing the targeting task with implicitly triggered techniques). Overall, 64% of all display crossings were performed with path or free, respectively, 27% with pointer warping, and 4% with the WIM control. Usage of implicitly triggered transitions was high for navigation between wall displays (86% on average for adjacent displays in T1-T3 and 75% for jumping between the outermost wall displays in T4). Implicitly triggered transitions were also the main choice for navigation between monitor and wall displays in T5-T7 (64%). However, pointer warping was employed more often than path or free whenever the tabletop display was involved, *i.e.*, T8 and T9 (40% and 55%, respectively). Usage frequencies are visualized in Figure 6.4.

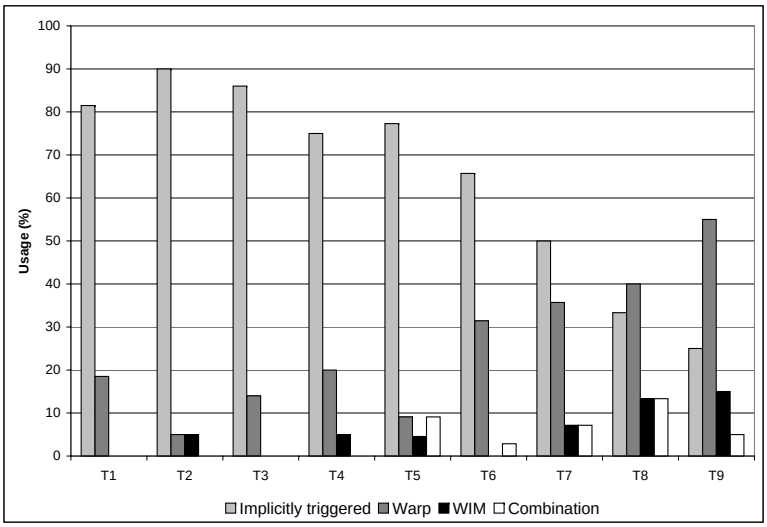


Figure 6.4: Percentage of usage for the implicitly triggered technique (path or free navigation, respectively), warp, WIM, and combinations in each transition.

Subjective Data

From the post-experiment questionnaire, we found no significant differences across preference scores using a one-way repeated measures ANOVA ($F_{3,54} = 3.241, p = .461$). On average, pointer warping was rated highest, as illustrated in Figure 6.5. In the interview, most users mentioned they preferred the combined condition.

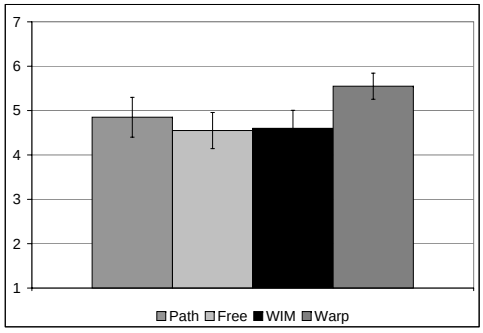


Figure 6.5: Subjective ranking of the four navigation techniques on a 7-point Likert scale.

In an interview at the end of the experiment, three users reported they preferred path navigation over free navigation due to the given structure implied by the constraint, visualized paths, while three other users assessed path navigation as "exhausting" caused by the restrictive paths. Users mentioned too small paths and paths to distinct displays being located too close together on the same display edge as design problems of path navigation.

Feedback for free navigation was similarly diverse: seven users rated free navigation as

very intuitive while two users stated they did not understand the concept of free navigation at all. The main point of criticism was the mouse mapping on the table, leading to targeting difficulties on the tabletop display. Additionally, users criticized the "irregular mouse pointer speed" across the displays, and five user mentioned that they lost their mouse pointer frequently when using free navigation.

For the WIM control, users reported that the start-up latency was too high and that "too many mental steps" were involved. Some users also indicated that the window placement at the mouse pointer location was not appropriate. They argued that their focus was already on the target display when the WIM window would appear at the current pointer location, forcing them to look back and identify the target in the miniature view. Thus, some suggested to open the WIM consistently at the home monitor or synchronously on all displays.

Pointer warping was generally appreciated by the users as it was perceived as fast option to change displays. Even the non-expert users in our experiment were familiar with basic keyboard shortcuts. Six users reported that using keyboard shortcuts for redirecting the mouse input was "intuitive". Display IDs were assigned in a circular manner (*cf.*, Figure 5.9), so users could easily remember the shortcut numbers. Users also reported that pointer warping was convenient when loosing track of the mouse pointer, as it would consistently warp the mouse back to a pre-defined location. Considering usage frequencies, these reports are surprising, as pointer warping was less often employed than implicitly triggered transitions. When asking participants for their strategy when to employ pointer warping or the WIM control instead of implicitly triggered transitions, they mentioned long distance travels, "not easily accessible" displays, and tabletop display and monitor in particular.

Observational Data

To explain some of the differences in task completion times and usage frequencies, we analyzed the video-taped sessions and also took field notes during the experiment. Additionally, we created movement plots on demand to better visualize the mouse movement trails. This observational data has to be considered as exploratory and requires verification in controlled experiments.

Path navigation was – on average – the fastest technique to move the mouse pointer across the three wall displays. However, completion time measures indicate a decreased performance whenever transitions are getting more complex in terms of depth change, non-opposite edge connections, and to reach displays partially outside the user's field of view. We observed several navigation problems in these situations: First, paths to adjacent displays were often located directly next to each other on a single display edge, causing users to pick the wrong path. In the video, we observed that users therefore frequently followed the mouse pointer with their gaze while finding the path to the target, in contrast to free navigation. Second, performance of path navigation suffered most when

transitioning to and from the tabletop display (T7 and T9). We assume that this is caused by the path connection between non-opposite edges – although an informal observation of *PointRight* [129] indicated that non-opposite edge connections do not cause any targeting difficulties. Third, varying distances between displays caused paths to have a small size on the more distant display, for instance on the wall projection for T6. This is caused by the global approach of pair-wise planar mapping, which does not take the user location into account. Ha *et al.* [97] already demonstrated that users would change path placements considerably depending on their location towards the displays.

Free navigation had better performance than path navigation when transitioning from the monitor or tabletop display to a wall display (T7 and T9). On average, performance was also higher for T5. What these transitions have in common is that source and target displays are located at varying distance to the user. T7 and T9 additionally required a crossing of one intermediate display. In contrast to path navigation, we rarely observed extensive path-searching activities by the users in the video analysis of these transitions. Users rather fluently moved the mouse towards the target. Targeting performance on the wall displays (T1-T4) is slightly slower compared to path navigation – probably due to the perspective C/D gain adjustment. Mouse pointer speed on the wall displays was comparably slow, so users sometimes were required to "clutch" the mouse for long traversals.

Targeting problems on the tabletop display most likely caused the low performance of free navigation in T6 and T8. Although T6 did not involve transitioning to the table, some users accidentally missed the monitor and reached the table instead. The number of unwanted transitions, *i.e.*, involuntary navigation to the wrong display, was much higher than with path navigation. We speculate that loosing the pointer depends on the display-less space between two displays in the perspective map, as the outcome position on a distant display would vary significantly if the movement direction was slightly modified. The low performance when steering on the tabletop display (T8) was probably caused by the C/D gain adjustment on the static perspective map. On the table, the mouse pointer navigation space was skewed and rotated approximately 90° compared to the other techniques using standard C/D gain. This was helpful in some situations, for instance when navigating away from the table (see performance of T9), but caused problems when selecting a target on the table itself. Video analysis revealed that almost all users turned head, body, and sometimes even the mouse itself when selecting a target on the table. Thus, the perspective map did not match the field of view any more, which is known to have a negative impact on navigation performance [265]. In addition, the viewing space did not correspond to the navigation space in free navigation, as the content was not compensated for the user's estimated viewing plane. This effect is known to cause steering difficulties [169]. Some users could accommodate for this situation easily, while others discovered serious problems and did not understand how mouse movement was interpreted by the system. This probably explains the high variance of task completion times on the tabletop display.

The slow task completion times of the WIM are obvious in the results and were more

closely analyzed in the videos. We identified two reasons for latencies: First, system latency is introduced, as the WIM control requires a window to be opened and desktop images of all machines to be streamed via network. The start-up latency takes approximately one second. Even with system latency subtracted from the task completion times, the WIM control was still the slowest technique. Second, subsequent to WIM window appearance, users required a short orientation phase. In the video, we could identify people searching targets in the environment, then shifting attention back to the display where the mouse pointer was located and the WIM would appear. This interaction pattern explains, for instance, why the WIM was the only navigation technique performing worse for moving *from* the table (T9) compared to transitioning *to* the table (T8), as vast head movements were required between the left projection wall and the tabletop display. This behavior is a bit surprising, as the WIM actually showed live desktop textures, thus the target was visible in the miniature view. Nevertheless only a small number of users was looking for targets appearing in the WIM.

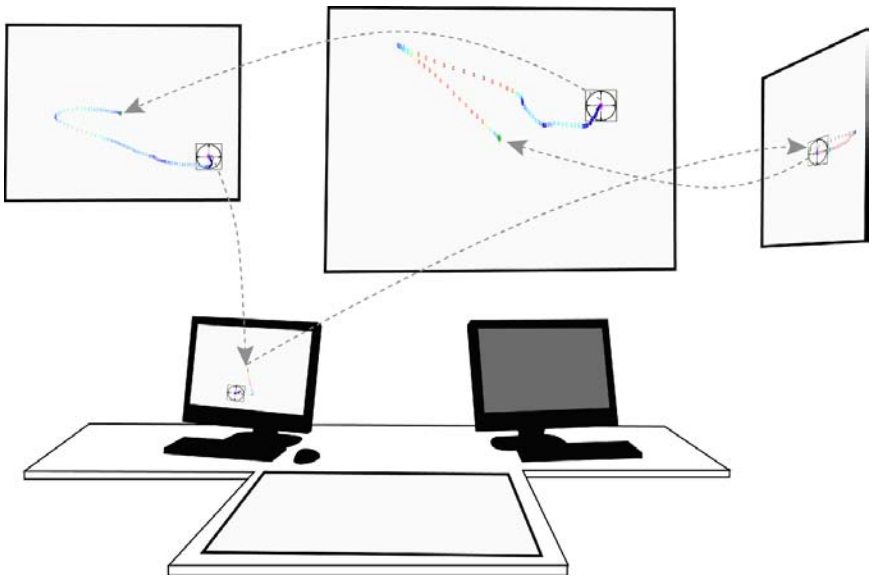


Figure 6.6: Movement sequence in the warp condition of user 8: starting from the monitor, the user selects a target on each wall display and then returns to the monitor. Movement trails between centered outcome position and target are color-coding movement velocity.

Although pointer warping was the fastest technique overall, it was not faster for target selection on the wall displays. Fitts' Law would suggest a lower task completion time in all transitions, as the effective travel distance in device space is higher using implicitly triggered navigation techniques that require the user to move the mouse pointer across all visible pixels. In the video analysis, we made the following observation: Even though the keyboard shortcut to trigger pointer warping could be accomplished with a single hand and most users (except for some non-expert users) did not look to the keyboard, there was a short delay introduced by pressing the shortcut. Additionally, most users

tended to employ an initial movement *away* from the target if the target was located in between the start position and the outcome position after the pointer warping operation. From the video analysis, we presumed that this "overshooting" behavior is caused by two factors: First, the initial mouse movement is continued in the direction of the pointer warp. Second, the length and pointer speed of this initial movement is influenced by the physical distance that has been warped. The larger this distance the more extensive the overshooting behavior. Figure 6.6 shows movement plots recorded for one user during the study, mapped onto the respective displays, which illustrate these observations. This observation suggests that the spatial display factors do have an effect on the movement quality and performance of pointer warping. To verify these observations, we conducted a controlled follow-up experiment to isolate the display factors and navigation techniques in question (Section 6.2).

6.1.6 Discussion

Subsequently, the findings and observations of the experiment, as well as implications and future research directions, will be discussed with respect to the design space of navigation techniques presented in Section 5.1.

Trigger

On the one hand, path navigation was rated as too restrictive by some users as the display-connecting paths were perceived as too small. On the other hand, the ability to leave the display at any display edge position was a major problem for free navigation, as participants often lost their mouse pointer when they involuntarily touched the display edge and thereby caused a transition to a remote display. The ideal solution would be to predict whether the user is actually intending to leave the display by analyzing the motion pattern. Thereby, we could preserve the display edges as valuable navigation aid when selecting items located at the boundaries of the displays [5], such as menu bars, while letting the user navigate quickly across display borders.

Our study suggests that triggering input redirection through pointer warping techniques leads to increased performance and is appreciated by participants across all experience levels. However, it also indicates that users rarely employ pointer warping when they have the choice. In fact, they choose pointer warping primarily to overcome subjectively complex transitions where extensive movement planning, physical effort, or adaptations of pointer movement directions (as for the tabletop) would be required when using seamless navigation techniques. Previous investigations have shown that pointer warping is beneficial when crossing multiple homogeneous monitors [28], accessing heterogeneous displays with strong size-resolution mismatches [29], and when sitting at an inconvenient location towards the displays [257]. From our results, we additionally speculate the pointer warping is superior for accessing remote displays with depth offsets or orientation changes. Thus, we recommend providing pointer warping techniques as additional option when building

mouse-controlled MDEs, so users can overcome complex display crossings and quickly relocate their pointer to a known position.

Explicitly triggering a transition in a world-in-miniature view did not indicate any improved performance as compared to a shortcut-triggered pointer warping. Also, it was mostly slower than implicitly triggered input redirection. Biehl and Bailey [34] found similar evidence when comparing their iconic miniature view to implicitly triggered input redirection in an MDE. However, they demonstrated a benefit of the WIM when using it for relocating application windows.

Cross-Display Movement

We cannot derive any implications or future research directions from quantitative results of our experiment, as none of our navigation techniques allowed movement in display-less space. However, based on some observational results, continuous movement in display-less space may be a solution to decrease the frequency of pointer losses in free navigation. Users sometimes unintentionally touched a display edge, which immediately caused a transition to a remote display, which was placed at a potentially large distance from the source display. The more distant the remote display was on the perspective map, the more deviation a slight movement direction change would cause on the outcome position when trying to move the cursor back. Continuous movement in display-less space has also been found to increase targeting performance when traversing small display gaps [18, 170]. However, with increasing gap size, warping the pointer across display-less space was found to improve performance [168].

Outcome Position

We observed that users sometimes had difficulties spotting the mouse pointer after performing a transition. This was partly caused by a network delay when performing input redirection. We did provide the users with an animated dot to signal the outcome position on the remote display. However, when the target display was not in their field of view, they did not have a visual cue about the current mouse pointer location. It is worth investigating whether more sophisticated visual cues indicating the current mouse pointer location (*e.g.*, a technique like *Anchored Cursor* [195]) help the users finding their mouse pointer more easily. However, it is also important to find out whether more obtrusive visual cues interfere with collaborative work in a group.

An important aspect in this context that has not been covered by our experiment is transitioning between overlapping displays. As an example, a monitor or tablet PC placed on a tabletop projection would lead to an outcome position in the inner region of the tabletop display, *i.e.*, not on the display edge. Similarly, a monitor partially occluding a wall projection would lead to a non-peripheral outcome position on the wall display. Display overlaps cannot be properly modeled with pair-wise planar mappings, where display connections are limited to edge regions. Overlapping displays therefore in part explain why

perspective-based navigation had a much higher performance compared to conventional stitching in an earlier study by Nacenta *et al.* [170].

C/D Gain

The perspective C/D gain adjustment applied by free navigation caused serious navigation problems, such as targeting difficulties on the tabletop display, as well as low mouse pointer speed on the wall displays. While it seems useful to have a perspective representation of the environment to determine the outcome position on the remote display, having a perspective C/D gain does not seem to bring any advantage. In contrast, users reported that the mouse was too slow on large, distant displays and that navigation on the table was unintuitive using a perspective C/D gain. Mind, however, that in our experiment, the viewing space differed from the navigation space – a combination known to have a negative impact on steering behavior [169]. In the future, we will therefore investigate different combinations of navigation parameters, combining positive aspects of path and free navigation.

6.2 Comparison of Pointer Warping and Seamless Navigation

In the previous experiment, we discovered evidence that pointer warping is overall the most efficient technique for cross-display navigation. Its performance seems to be only slightly influenced by emerging display factors in contrast to implicitly triggered, seamless navigation techniques. However, detailed movement analysis suggests that pointer warping suffers from different targeting problems, so the performance is lower in homogeneous settings. To better understand these phenomenons, we analyzed the inherent differences to standard cross-display mouse behavior, which may lead to differences in performance results:

- Instead of moving the mouse pointer continuously across a display edge, the user is required to explicitly trigger the pointer warping operation, for instance by pressing a mouse button or keyboard shortcut. While this potentially minimizes the required movement planning, pressing the trigger may introduce an additional mental and physical overhead.

- Pointer warping usually minimizes the required pointer travel distance in device space – and thereby the index of difficulty (ID) described in Fitts' Law [144] (Equation 2.1) – at the expense of an increased visual-device space mismatch [29], *i.e.*, the disruption between the visually perceived path between start and target location and the actual device space.

- The pointer warping operation can be initiated from any display location, leading to a dynamically changing visual-device space mismatch, despite a static display setup.

- Depending on the outcome position on the target display, targets may lie in between start and outcome position, necessitating the user to correct the pointer movement direction after performing the warp. Alternative outcome placement strategies (*e.g.*, [28, 29]) or facilitating additional information from head-tracking (*e.g.*, [7, 28]) can minimize this effect depending on the desired application.

The properties of pointer warping are also illustrated in Figure 6.7.

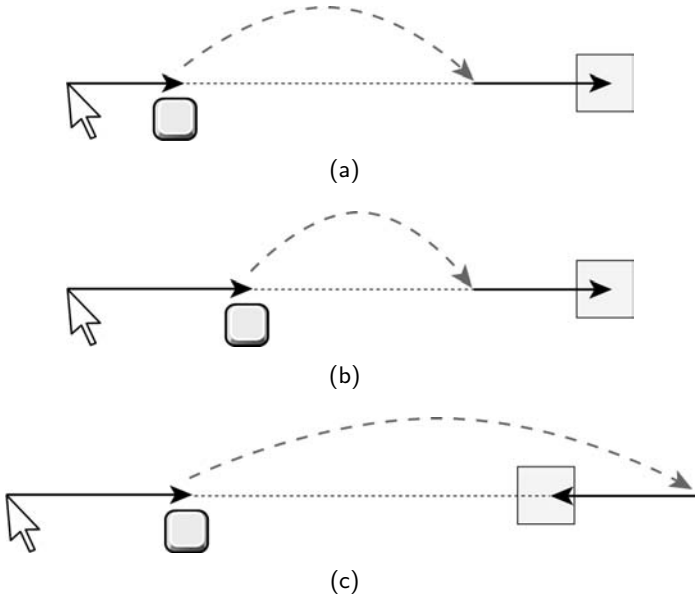


Figure 6.7: Properties of pointer warping: (a) it has to be explicitly triggered and reduces the travel distance in device space, (b) the visual-device space mismatch dynamically changes, depending on the trigger location, and (c) targets may lie in between start and outcome position.

In the previous experiment, we observed that users tended to make an initial movement away from the target after warping the pointer in certain situations. According to our observations, this movement was influenced by two factors:

1. the pointer warp direction, which influences the initial pointer movement direction irrespective of the actual target location and

2. the distance of the warp, which influences the length of this initial movement.

6.2.1 Research Questions

Based on this analysis and our previous observations, we formulated two research questions concerning the effect of spatial display arrangements on pointer warping behavior:

***Q1**: Does an increased visual distance affect pointer warping differently than standard mouse behavior?*

Nacenta *et al.* [168] showed that with a large physical distance between monitors, minimizing the ID by warping the mouse pointer across the gap outweighs advantages of minimizing the visual-device space mismatch by using a mouse ether [18]. However, they also discovered a slight performance loss for warping the mouse across the gap, which they explain with extended movement planning periods due to the visual-device space mismatch, and target overshooting. According to our observations in the previous experiment, visual distance also has an effect on the amount of overshooting with pointer warping. However, as pointer warping does not require a continuous movement – and therefore less movement planning – it may be expected that overshooting will be less distinct compared to standard mouse behavior. We aim to evaluate the impact of visual-device space mismatch on pointer warping by changing the physical distance between adjacent monitors and comparing the effects with standard mouse behavior.

***Q2**: Are targets between start and outcome position harder to reach?*
Pointer warping might relocate the mouse pointer "farther" than the anticipated target location (Figure 6.7(c)). With respect to the direction of the pointer warp, users therefore may have to re-adjust the pointer movement direction after performing the warp. To our knowledge, this aspect has never been investigated before and has been observed when warping the pointer in a heterogeneous MDE (Section 6.1). To evaluate this effect, we compare task times of targets located before, on, and after the outcome position relative to the start position, and evaluate overshooting characteristics.

6.2.2 Participants

Fifteen right-handed participants (13 male, 2 female, aged 25 to 37) attended the experiment. All users, except for one, work with two or more monitors on a regular basis.

6.2.3 Navigation Techniques

As navigation techniques, we employed standard mouse behavior (*mouse*) and pointer warping (*warp*). In the mouse technique, the inner monitor edges are connected in device space (as illustrated in Figure 4.13(a)) and the mouse pointer is warped across display-less space after implicitly triggering a transition. In the warp technique, a transition to the adjacent display had to be explicitly triggered by pressing the space bar. The outcome position was set to the center of the target display. Center placement is not necessarily the most adequate placement strategy for many tasks [28, 29]. However, we chose this placement for our experiment as it keeps the outcome position consistent and is therefore easier to compare across the experimental conditions. Furthermore, the fundamental characteristics of pointer warping with respect to our research questions are not affected by this simple outcome placement.

6.2.4 Setup

The experiment was conducted on a homogeneous dual-monitor setup consisting of two identical 22" wide-screen monitors (1680x1050 pixels). The distance between the two monitors was varied. In the *near* condition, the two monitors were placed directly next to each other, separated only by a 3.5 cm monitor bezel. In the *far* condition, the display-less space between the monitors (including bezels) was approximately the width of one monitor. In both conditions, the user was sitting in front of the left monitor. Figure 6.8 illustrates the setup.

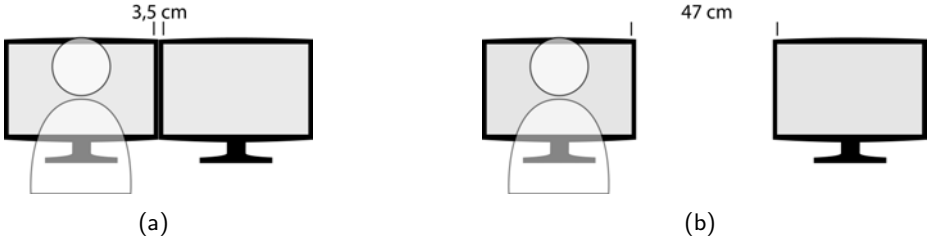


Figure 6.8: The monitor setup in the experiment in the (a) near and (b) far condition.

Mind that changing the monitor distance only altered the physical setup. The device spaces of mouse and warp were unaffected.

6.2.5 Task

Users were asked to press alternating 50x50 pixels start and target buttons. Start buttons were always located on the left (source) display, target buttons on the right (target) display, so the experiment was limited to one movement direction. Start and target buttons were distributed to five locations on source and target display, respectively: left top (LT), left bottom (LB), center (C), right top (RT), and right bottom (RB), resulting in possible 25 cross-display movements. Each target location was separately analyzed as movement path (combined from five start locations), as illustrated for LT in Figure 6.9. Note that the path to center (C) could be accomplished with warp without moving the mouse, as the target C was located at the outcome position of the mouse cursor after the warp. The left

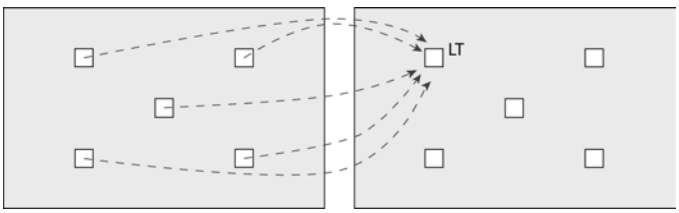


Figure 6.9: The five potential start locations and the target locations on the right display. For each target location (as in this example LT) performance measures of the five start locations were accumulated.

targets (LT and LB) were located in between the source display and the outcome position after warping. LT, LB, RT, and RB have the same ID on the target display for warp (with center placement).

6.2.6 Design and Procedure

The study followed a 2x2x5 within-subjects factorial design with the following factors:
navigation technique: mouse and warp,
monitor distance: near and far, and
path: LT, LB, C, RT, and RB.

Besides the task time between start and target button selections, we evaluated activity measures, such as the *time spent on the source display*, which indicates extended orientation or planning periods, as well as *distance traveled* for source and target display, respectively. For an optimal target-selection task with pointer warping, the movement distance and time spent on the source display is 0. Furthermore, we analyzed the amount of overshooting by defining a task axis [145] on the target display – from the first position the mouse pointer appears on the target display to the center of the target. We discriminate two overshooting measurements: classic *target overshooting*, and *entry overshooting*, *i.e.*, the amount of pointer movement away from the target after warping the pointer (Figure 6.10).

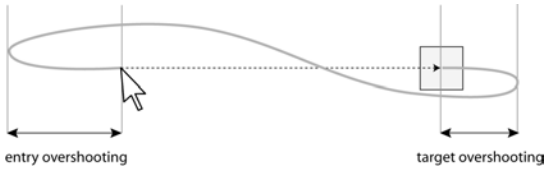


Figure 6.10: Task axis connecting pointer outcome position and target position on the target display with entry and target overshooting.

Mind that the aim of the experiment is not to directly compare the performance of mouse and warp but rather to analyze the differences in performance and movement characteristics implied by a varying visual-device space mismatch and movement paths.

Each participant had to accomplish four runs with 25 cross-display path sequences. 25 path sequences on the left monitor were added to prevent users from immediately switching the monitor after clicking the start button, but were not evaluated. The order of navigation technique and monitor distance, as well as path sequences, was balanced. At the end of the experiment, users had to subjectively assess the two navigation techniques for the two monitor distances. Afterwards, a semi-structured interview was conducted.

6.2.7 Results

Apart from performance measures (*i.e.*, completion time between pressing start and target button), we additionally logged all mouse movement and keyboard events. Data was logged

at a maximum frequency of 125 data points per second. Accuracy measures – like entry overshooting, target overshooting, distance traveled, and time spent on display – were extracted from these logs. All measures were evaluated using a 2 (navigation technique) x 2 (monitor distance) x 5 (path) repeated measures ANOVA with $\alpha = .05$ for main effects and interactions and Bonferroni adjustments for post-hoc comparisons. Results are summarized in Table 6.2.

	df	*F*					
		CT	EO	TO	DT	DS	TS
N	$(1, 74)$	23.033**	–	44.6**	101.1**	208.7**	37.5**
D	$(1, 74)$	118.745**	2.1	81.1**	118.9**	7.1**	38.2**
P	$(4, 269)$	42.664**	56.3**	12.6**	82.5**	2.1	7.4**
N*D	$(1, 74)$	11.151*	–	78.2**	65.2**	17.8**	4.8*
N*P	$(4, 269)$	23.953**	–	52.7**	113.8**	1.5	14.5**
D*P	$(4, 269)$	2.562*	0.6	26.2**	31.7**	0.5	2.8*
N*D*P	$(4, 269)$	2.585*	–	28.9**	33.4**	0.9	0.7

Table 6.2: ANOVA for (**$p < .001$, *$p < .05$) completion time (CT), entry overshooting (EO), target overshooting (TO), distance traveled on target display (DT), distance traveled on source display (DS), and time spent on source display (TS). Main effects for navigation technique (N), distance (D), path (P) and interactions are shown.

Effect of Monitor Distance

Post-hoc comparisons of completion time (Figure 6.11) show that both, mouse and warp, were significantly faster with monitors near than with monitors far ($\Delta t_m = 289.6$ and $\Delta t_w = 167.5$). However, the effect on mouse by the changing physical gap seems to be stronger: while mouse was faster than warp with monitors near ($t_m = 1435.1$ and $t_w = 1621.3$), there is no statistically significant difference between mouse and warp with monitors far ($t_m = 1724.6$ and $t_w = 1788.8$).

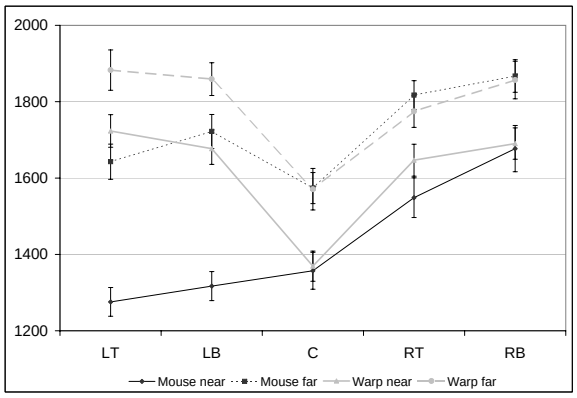


Figure 6.11: Average task completion times (ms) and standard error of mouse and warp in near and far.

Participants did not like having the monitors spaced apart, as they had to turn their head to see the target. But with increasing distance between the monitors, they started to

appreciate pointer warping: a two-factorial ANOVA of seven-point Likert scale ratings for mouse and warp on near and far, respectively, revealed an interaction between navigation technique and distance ($F_{1,14} = 17.148, p = .001$). Mouse was evaluated higher for monitors near, but there was no difference in the ratings for monitors far. Users mentioned that they felt like they *"had to move the mouse farther"* with monitors far and that *"the mouse was too slow"*, whereas with pointer warping they *"always knew where the mouse was located"* after the warp. One user stated: *"as the monitors were no longer spatially connected, the mouse pointer path was not intuitive"*.

Effect of Target Location

As expected from Fitts' Law, left targets (LT, LB) could be selected fastest with mouse ($t_{LT} = 1459.2$ and $t_{LB} = 1519.5$) and center was selected fastest with warp ($t_C = 1470.0$). With warp, RT was selected significantly faster than LT ($t_{RT} = 1710.7$ and $t_{LT} = 1803.0$) – despite equal ID. This difference partially confirms that targets located between start and outcome position (*i.e.*, LT and LB) are harder to reach with warp.

Overshooting

Target overshooting was significantly higher for mouse (57.2 px) than for warp (22.8 px). For mouse, target overshooting was higher for the left targets (108.3 px). In contrast, for warp, target overshooting was highest for the right targets (47.5 px). As also observed by Nacenta *et al.* [168], target overshooting for mouse was higher with monitors far (94.6 px) than with monitors near (19.8 px). For warp, there is no target overshooting difference between near and far (22.1 px and 23.5 px, respectively).

We also measured entry overshooting for warp: For the left targets, there was significantly more entry overshooting (157.8 px) than for the right targets (2.1 px). However, there was no main effect of entry overshooting for distance (89.2 px for near and 101.8 px for far, respectively). All users in our experiment were aware of entry overshooting in warp and most could recall that the initial movement direction was towards the right. All users stated that this movement was performed unconsciously and that it was somehow annoying. Figure 6.12 shows a sample plot of one user's movement trails to illustrate the effect of entry overshooting.

Activity Measures

For mouse, there was more distance traveled on both, source and target display, than for warp. This is not surprising, as the ID for warp was much lower in our experiment than for mouse. However, the time spent on the source display was significantly higher for warp (524.8 ms) than for mouse (448.3 ms) – although there was actually no movement required on the source display for warp. For both navigation techniques, there was more

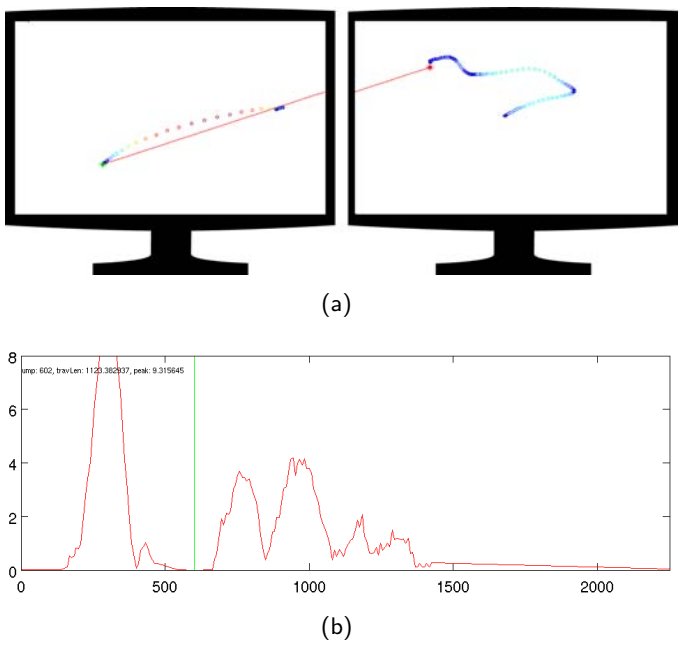


Figure 6.12: Example movement trail of user 5 for pointer warping with monitors near and path RB-LT: (a) The red line shows the optimal route in device space. The user started on the left monitor and first quickly steered towards the target before explicitly triggering input redirection. On the target display, the pointer is set to the center of the display and the user initiates the pointer movement in the warp direction before correcting the movement. (b) The red trail shows the velocity (px/ms) over time (ms). The green line indicates the time of the warp trigger.

time spent on the source display in the far condition than with monitors near ($\Delta t_m = 109.3$ and $\Delta t_w = 61.1$).

6.2.8 Discussion

We will discuss the implications of our experiment based on our research questions:

Q1: *Does an increased visual distance affect pointer warping differently than standard mouse behavior?*

Increasing the physical gap between the monitors affected both, standard mouse behavior and pointer warping. However, the increase in task completion time was higher for standard mouse behavior (about 20%) than for pointer warping (about 10%). For both techniques, we could observe an extended initial non-movement period with monitors far compared to monitors near. This is an indication for the additional effort to turn the head to the distant monitor and find the target there. For standard mouse behavior, we additionally discovered increased target overshooting for the targets located near the left display boundary – an effect also observed by Nacenta *et al.* [168]. This extended overshooting can explain the decreasing performance for mouse in contrast to warp. A longer time spent on the source display despite a lower amount of pointer travel indicates that

pointer warping requires an extended planning period as compared to standard mouse behavior – irrespective whether the monitors are far or near.

Benko and Feiner [28] demonstrated a benefit of pointer warping for bridging long distances in the device space. Complementing their work, our experiment indicates an advantage of pointer warping for bridging gaps in the visual space with unchanged device space: If users do not perceive the visual space as continuous due to large physical gaps, standard mouse behavior leads to increased targeting problems so pointer warping achieves comparable performance and user acceptance.

Q1: *Are targets between start and outcome position harder to reach?*
Although pointer warping is not a continuous operation, the direction of the warp influences the subsequent pointer movements of the users: mouse pointer movement is first initiated in the direction of the warp and is then corrected towards the actual target location. This is reflected in the higher amount of target overshooting for the right targets and entry overshooting for the left targets with a warp direction from the left to the right monitor. Targets lying in between start and outcome location showed a slightly weaker targeting performance than those lying on the right (*i.e.*, the extension of the warp direction) or directly at the outcome position. The amount of this overshooting behavior is not related to the distance of the warp.

Due to the noticeable performance decrease and subjective annoyance by the users, situations where the user has to re-adjust the movement direction after the warp should be avoided. Designers of MDEs should consider dynamic placement strategies taking into account the start location of the warp and areas with high probability of user interaction on the target display. By setting the warp outcome position between the intersecting display boundary of the warp and the closest interaction area, important interaction regions can be reached by continuing the warp movement direction, instead of causing a contrary direction adjustment (illustrated in Figure 6.13). Alternatively, additional information from head-tracking (*e.g.*, [8, 28]) or eye-tracking can help to select the optimal outcome position. However, tracking equipment can be rather obtrusive and is not always available.

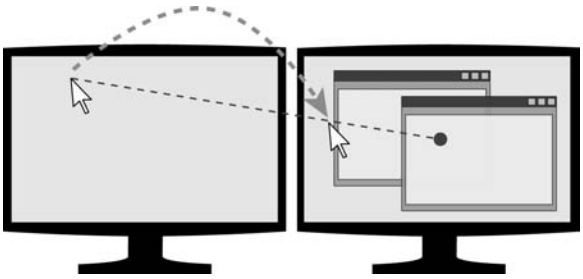


Figure 6.13: Suggestion for reducing entry overshooting: reducing the need to adjust the movement direction after the warp to reach important targets by taking screen content information into account.

With these results, we can partially explain the performance differences between pointer warping and implicitly triggered navigation techniques in the previous experiment

(Section 6.1): Displays located close together are perceived as continuous interaction space for the user, resulting in a higher subjective preference and higher targeting performance for implicitly triggered transitions. This explains the high usage frequencies and superior performance of implicitly triggered navigation techniques for wall displays in the previous experiment. However, increasing the visual-device space mismatch – in this experiment by increasing the display-less space – leads to a disruption of the continuously perceived interaction space. Users tend to increase target overshooting due to overestimation of the device space distance – an effect which has previously been observed by Nacenta *et al.* [168] – and show increased appreciation of pointer warping, which is less affected by physical discontinuities.

The entry overshooting behavior initially observed in the previous experiment could be replicated by analyzing the logged mouse events. However, the effect on pointer warping performance could only be partially verified. The higher target completion times for pointer warping rather stemmed from an increased planning period before initiating the explicit warping trigger. Also, we could not verify the effect of an increased visual-device space mismatch on the length of the entry overshooting movement.

6.3 Comparison of Outcome Positions for Pointer Warping

The previous experiments have shown that pointer warping is a highly efficient and also appreciated alternative to spatially aware, implicitly triggered cross-display navigation techniques. While implicitly triggered techniques are superior, in terms of performance, usage frequencies, and user satisfaction, if the spatial display arrangement is "intuitive", more complex display arrangements (*e.g.*, with large display-less space in between, depth offsets, or orientation changes) are obviously easier to handle with explicitly triggered pointer warping.

However, we showed that pointer warping is also affected by the spatial display arrangement. While it is rather insensitive to an increased gap between adjacent displays, the effectiveness of the movement on the target display depends on the movement direction of the warp in visual space. The most straight-forward outcome position on the center of the target display does not take this movement direction into account. Our research question that motivated this experiment therefore is: *Does incorporating the spatial display arrangement in the outcome position of pointer warping have an effect on targeting performance?*

We therefore designed an experiment to compare different outcome placement strategies for pointer warping.

6.3.1 Participants

Fifteen right-handed users (13 males, 2 females, aged 21 to 33) participated in the experiment. Twelve users are working with multi-monitor setups on a regular basis.

6.3.2 Outcome Placement Strategies

We compared three different outcome placement strategies:

Center, which has been used for the previous experiments, places the mouse pointer to the center of the adjacent display after triggering pointer warping. Thereby, it minimizes the travel distance to all potential target locations in device space.

Frame-relative sets the mouse pointer to the same relative position on the target display as the location where pointer warping has been triggered on the source display. In an experiment by Benko and Feiner [28], this placement was superior to center in a homogeneous multi-monitor setup. In contrast to center, mouse pointer movement on the source display has an effect on the outcome position on the target display [28]. However, it only takes the pointer travel in device space into account and disregards the spatial display arrangement, affecting the visual pointer travel. As a result, a pointer movement towards the target in visual space followed by the warping trigger may actually lead to a movement away from the target in device space (*e.g.*, moving the pointer towards the right on the source display followed by the warping trigger in Figure 6.14).

Border sets the outcome position of the pointer on the target display to the closest edge location from the starting point – with respect to the visual space. The entry overshooting effect observed in the previous experiment is thereby effectively avoided, as any initial movement in warping direction only moves the mouse pointer further along the visual path. Border warping is conceptually similar to spatially aware implicitly triggered navigation (especially free navigation) and only differs in the trigger parameter. Figure 6.14 shows all three outcome placement strategies.

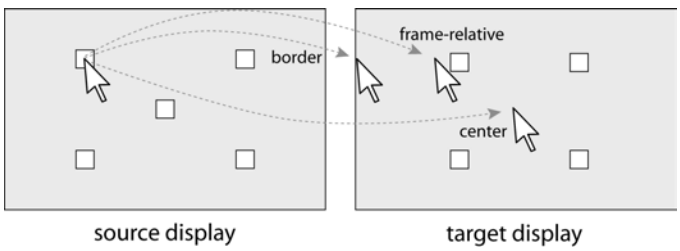


Figure 6.14: The three outcome placement strategies evaluated in the experiment: center, frame-relative, and border. Start and target buttons are illustrated on source and target display, respectively.

In a previous experiment, Benko and Feiner [29] also evaluated *frame-memory* placement, which warps the pointer to the last known location of the target display. They showed that frame-memory placement was superior to frame-relative placement and an implicitly triggered control condition. However, as frame-memory placement does not take the movement direction into account and has not been shown to be superior for homogeneous MDEs, we decided that frame-relative placement is the better comparison.

6.3.3 Setup

We employed the same dual monitor setup as for the previous experiment. However, all runs were conducted with monitors far apart – the distance where users felt equally comfortable using pointer warping and conventional mouse-based navigation.

6.3.4 Task

We employed the same targeting task as for the previous experiment, but only provided four targets (LT, LB, RT, RB) instead of five, as illustrated in Figure 6.14. This way, we can avoid situations where the mouse pointer appears directly on the target in the frame-relative and center conditions after warping. Mind that the average distance from the outcome position to the target – and, as a consequence, the index of difficulty [144] – varies for the three placement strategies: Center had the lowest average distance (351.9 px), followed by frame-relative (505.3 px). Border has an average distance to the left targets with 582.5 pixels and 1156.0 pixels for the right, respectively. For calculating the distance, we assumed the mouse pointer not to be moved on the source display and relied on the start button locations to calculate the outcome positions for frame-relative and border.

6.3.5 Design and Procedure

The study followed a 3x4 factorial design with the following two factors:
placement: center, border, relative, and
path: LT, LB, RT, RB.

The same performance and accuracy measures were used as for the previous experiment (Section 6.2). To fully compare the performance of those three techniques, we added the *throughput* measure [144], which relates the task completion time with the optimum travel distance and the target width, which has not been varied in our experiment.

6.3.6 Results

All measures were evaluated using a 3 (placement) x 4 (path) repeated measures ANOVA, with $\alpha = .05$ for main effects and interactions and Bonferroni adjustments for post-hoc comparisons.

Performance

Performance was measured by task completion time between start and target click, as well as throughput. For both measures, all main effects and interactions are significant (c.f. Table 6.3 and Figure 6.15).

Task completion time of center was significantly faster ($t_c = 1797.9ms$) than both, border ($t_b = 1870.3$) and frame-relative ($t_r = 1924.3$). Center was faster than border

for the right targets, but slower than border for LB. However, post-hoc comparisons of throughput showed that border had the highest throughput ($tp_b = 2.82$), center the lowest ($tp_c = 2.23$).

	df	*F*	
		Completion time	Throughput
Placement	$(2, 148)$	6.5*	103.1**
Path	$(3, 222)$	4.4*	15.9**
Placement*Path	$(6, 444)$	7.0**	6.1**

Table 6.3: ANOVA for task completion time and throughput (**$p < .001$, *$p < .05$).

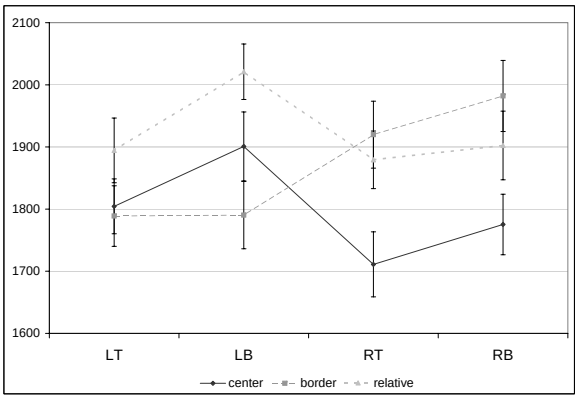


Figure 6.15: Average task completion times (ms) and standard error of the placement strategies center, border, and frame-relative for the four paths.

Accuracy Measures

Results for accuracy measures are summarized in Table 6.4.

	df	*F*					
		EO	TO	PV	DT	DS	TS
Placement	$(2, 148)$	40.0**	24.5**	14.0**	103.0**	3.3*	5.6*
Path	$(3, 222)$	30.7**	2.1	1.0	0.4	0.9	0.9
Place.*Path	$(6, 444)$	17.8**	12.1**	1.0	15.9**	0.8	0.3

Table 6.4: ANOVA for accuracy measures (**$p < .001$, *$p < .05$) entry overshooting (EO), target overshooting (TO), peak velocity (PV), distance traveled on target display (DT), distance traveled on source display (DS), and time spent on source display (TS).

There is no statistically significant difference between entry overshooting for the center and frame-relative placement (70.8 px and 57.8 px, respectively). Similar to the previous experiment, there is more entry overshooting for the left targets than for the right targets (110.7 px and 17.8 px, respectively) with frame-relative, as well as center.

There was significantly more target overshooting with border (67.9 px) than with both, center and frame-relative (17.9 px and 20.9 px, respectively). Target overshooting

for border occurred primarily at the left targets (110.3 px), while there is no difference between placements for the right targets (35.4 px, 25.5 px, and 25.9 px).

With border, users had significantly longer travel distances on the target display (1165.3 px) compared to center (631.5 px) and relative (781.0 px). Border also had slightly more travel on the source display (155.9 px vs. 113.4 px and 117.8 px). However, this difference is not statistically significant. The time spent on the source display was highest with relative (492.6 ms), which is significant with respect to border (422.9 px). The peak velocity reached was higher with border (1.25 px/ms) than with center (0.97 px/ms) or frame-relative (0.93 px/ms). In 91% of all cases, the peak velocity was reached on the target display.

Subjective Data

At the end of the experiment users were asked to evaluate the three placement strategies on a seven-point Likert scale. A repeated measures ANOVA analysis did not show any differences between ratings ($F_{2,28} = .942, p = .402$). On average, border was rated highest (5.3), frame-relative lowest (4.5).

In a subsequent interview, seven users expressed a clear preference for border, as it was more similar to conventional cross-display mouse interaction and necessity for pointer searching on the target display was reduced. Three users clearly preferred center and two users clearly preferred frame-relative over border, because of border's long travel distances. Three users were undecided about their favored placement strategy.

6.3.7 Discussion

We will discuss the implications of our experiment with respect to the research question formulated above: *Does incorporating the spatial display arrangement in the outcome position of pointer warping have an effect on targeting performance?*

We introduced a new pointer warping outcome placement strategy that incorporated basic spatial knowledge of the environment to overcome targeting problems caused by the visual-device space mismatch. The decrease of the visual-device space mismatch was introduced with the cost of a longer travel distance in device space and, as a consequence, a higher index of difficulty.

Our experiment showed that this spatial knowledge did not increase performance. In contrast, the naïve center placement, which minimizes the index of difficulty in the device space, outperformed both, border and relative placement in terms of task completion time. However, center had the lowest throughput and border placement the highest.

Accuracy measures revealed that border warping caused similar navigation problems as implicitly triggered navigation in our previous experiment: peak velocity was higher than for the other placement strategies, which is probably responsible for the significantly increased amount of target overshooting for the left targets. Although we cannot directly compare implicitly triggered navigation with border warping, we can assume that border

warping also suffered from inherent pointer warping problems, such as an increased non-movement period before triggering the warp, which adds a constant overhead.

One additional aspect worth discussing is the diverging result from the experiment of M^3 [28], where frame-relative outperformed center placement. A difference may be caused by placement of targets. In the experiment by Benko and Feiner [28], the target was always located in between the outcome position of frame-relative and center to ensure equal travel distances. If the start position was kept at the outermost display boundary, as suggested by the experimental design description, the target could always be reached by extending the warp direction in frame-relative, while a 180° movement correction was required in center – most likely leading to our observed entry overshooting and thereby limiting performance. We rather tried to evenly distribute the start and target locations, leading to a clear advantage of center placement, which minimizes the travel distance to all potential target locations. For a fair comparison, we therefore included the throughput measure to relate task completion time with the index of difficulty. Indeed, frame-relative also had a higher throughput than center in our experiment.

6.4 Discussion

Based on the results of our three multi-display navigation experiments, we can confirm that incorporating knowledge of the environment for cross-display mouse pointer navigation techniques helps users to *access* information in "simple" MDE settings. Exploratory evidence and subjective feedback suggests that users perceive cross-display navigation as *"complex"* if intermediate displays have to be crossed, if a large display-less space has to be bridged and therefore the displays seem to be *"no longer spatially connected"*, a smaller display which is located closer to the user is targeted, and when steering towards a display which is rotated along two axes compared to the source display. In these cases, performance of implicitly triggered (and thus spatially aware) techniques significantly drops, while performance of explicitly triggered techniques is hardly affected. We could verify the effect of gaps between adjacent displays on the performance of implicitly and explicitly triggered techniques: pointer warping was hardly affected by an increasing display-less space.

By analyzing the movement trajectories in targeting tasks, we could verify that the spatial display arrangement does have an effect on the pointer movement – even when using simple pointer warping with center placement. However, any attempt to compensate for negative side effects, such as the observed entry overshooting for pointer warping, by decreasing the visual-device space mismatch, increases the index of difficulty in device space. Our spatially aware border warping strategy, that takes the warping movement in visual space into account, lead to an increased throughput, but could not increase overall targeting performance, compared to simple center placement with minimal ID.

From our observations, we could derive some implications for multi-display navigation design and recommendations for practitioners:

Provide explicitly triggered input redirection as alternative.
Implicitly triggered input redirection can easily be provided as additionally available alternative to implicitly triggered cross-display navigation. Especially for frequent paths, for instance navigation back to the home display (like in transition T6 of the first experiment), an easy shortcut could be provided. It supports users in overcoming long and subjectively complex paths and also helps them to visually re-acquire their pointer. Empirical evidence has shown that pointer warping provides a benefit to overcome long distances in device space [28] due to its decreased index of difficulty. We could additionally show that pointer warping is beneficial to overcome long distances in visual space, despite short distances in device space. However, pointer warping adds a constant overhead – probably caused by the explicit trigger – which makes implicitly triggered and spatially consistent input redirection the primary choice for many display configurations.

Compensating for visual-device space mismatches does not necessarily lead to increased targeting performance.
Previous experiments have shown that minimizing the visual-device space mismatch by compensating for discontinuities caused by monitor bezels and size-resolution mismatches supported users in acquiring targets on a remote display [18, 170]. However, Nacenta *et al.* [168] later showed that the advantage of visual-device space compensation by applying continuous cross-display movement in the "ether" does not outweigh the increased ID in device space, if the gap is large. We additionally showed that visual-device space compensation by incorporating spatial knowledge into the outcome placement strategy of pointer warping does not compensate for the increased ID, in terms of task completion times. However, our results show a tendency towards a slightly higher user acceptance and a better throughput.

So far, we have evaluated cross-display navigation techniques only in isolation. We did not consider cross-display navigation for remote information access in a wider context, *i.e.*, embedded into more complex information management tasks in MDEs. In the future, it will therefore be necessary to assess the suitability of different navigation techniques in longitudinal experiments, where users have to accomplish more complex information management activities.

Part IV

Window Management for Emerging Display Environments

Chapter 7

Window Manager Extensions

When designing novel window management techniques for emerging display environments, fundamental window manager functionality has to be enhanced. It is no longer sufficient to consider the environment as simple rectangular screen, operated by a single user with a single mouse. In addition, treating windows as rectangular screen entities furthermore limits the potential window manager functionality.

In this chapter, extensions to the Compiz 3D compositing window manager (Section 2.3.1) to enhance the window manager for environment and window content awareness are presented. In particular, four directions are being discussed:

- Basic multi-user functionality (Section 7.1),
- two different methods how to access window content in the window manager (Section 7.2),
- awareness of and adaptivity to irregularly shaped displays (Section 7.3), and
- importance-driven compositing window management, a new approach to compositing window management facilitating knowledge of display form factors and window content (Section 7.4).

7.1 Multi-User Interaction and Identification

Multi-pointer X (MPX) [115] provides multiple independent mouse pointers and keyboard foci on windowing system level for the X Window System. Although MPX is already available in the most recent X Window System releases, application support is still scarce and most window managers do not reliably render or handle multiple pointers. We thus extended the Compiz window manager with basic multi-pointer support for our purposes. We implemented a plug-in that queries the *XInput2* event loop (a new X event loop for multi-pointer events) and renders each mouse pointer in its associated color.

MPX allows multiple users to interact concurrently with distinct applications, but it cannot resolve situations where multiple pointers attempt to access the same application concurrently, if the application itself is not multi-pointer aware. In principle, resolving multi-user input on application level is not the duty of a window manager. However, adapting existing applications – but also newly created applications – to multi-user input is challenging, as most GUI toolkits do not yet provide multi-pointer event notification. Thus, making applications multi-pointer aware requires substantial modifications and bypassing the GUI toolkit.

We therefore provide the possibility to query user identities based on events received in the application, such as keystrokes or button presses. The window manager establishes an interaction history for received input events (*i.e.*, mouse motion, button press, and key press and a corresponding time stamp). Applications connect to the window manager via the Deskotheque network interface, which matches event reports by applications with the plug-in's interaction history and returns the ID of the pointing device.

This window manager extension is facilitated by the central application coordination infrastructure presented in the next section (Section 7.2.1), but also by cross-display navigation applications, such as the world-in-miniature control (Section 5.3.3), to identify the user issuing an explicit input redirection trigger.

7.2 Accessing Window Content

Access to the window content is required whenever the window manager aims to provide content-aware information presentation or interaction techniques. We will discuss two approaches for accessing the content of the individual windows:

1. Applications are modified in a minimally invasive manner to communicate with a central application coordination routine (Section 7.2.1).

2. Window textures are evaluated by using a bottom-up visual attention model to identify "important" window regions (Section 7.2.2).

While the first approach is much more powerful, as applications can provide semantic information, the second approach can be implemented directly in the window manager, without modifying the applications. However, importance can only be estimated based on visual content features.

7.2.1 Central Application Coordination

To coordinate multiple applications and their content, we use an approach similar to *Snap-Together Visualization* [174], where multiple applications communicate via a lightweight API based on COM. Snapped applications provide synchronized mechanisms, such as "load", "select" (synchronized highlighting), and synchronized scrolling. Windows are coordinated pairwise based on a set of actions and join relationships.

In contrast to Snap, our approach uses a central management application, which coordinates user selections in multiple applications based on HTTP. The management application is implemented as Tomcat Java application server. In this way, a wide variety of applications can directly communicate with the management application, like web browsers, document editors, or visualization toolkits. As another distinguishing aspect to Snap, our central application coordination approach currently only supports synchronized highlighting. However, the concept can be easily extended to support more sophisticated synchronized activities, such as loading of shared data sets, viewport synchronization, or mechanisms to restore previous states of the application – as far as supported by the respective applications. In the remainder of this section, synchronized highlighting will be presented.

Synchronizing User Selections

Multiple client applications can register to the management process and report selection identifiers upon local user selection. The registered applications themselves determine how a user selection is triggered. Possible selection events are: hovering the mouse pointer over an element, marking text, or entering a search string. The selection is sent to the management application as character string – the *selection ID*.

After receiving a selection ID, the management application consults the window manager's interaction history (*cf.*, Section 7.1) to determine the most recently active pointer within the application window's boundary – *i.e.*, to find the user who triggered the selection. The selection ID is then distributed to all registered applications. Each client application evaluates the incoming selection ID individually. For instance, in text documents, these selection identifiers correspond directly to a substring of the document (*e.g.*, single words), while in a spreadsheet application it may be mapped to a whole column of a table with the selection identifier as column headline. To resolve more complex relations, client applications are free to map the incoming selection ID to others. For instance, an application may translate the incoming identifier string so it matches the language of the displayed data set or can determine a suitable hierarchy level for mapping hierarchical data structures. Once a client application determined entities matching the provided selection ID, it reports their bounding rectangles in the application window, or alternatively, single points, to the management application. Figure 7.1 illustrates these processes.

The current system infrastructure, as illustrated in Figure 7.1, is limited to a single host. However, all inter-component communication facilitates network interfaces, so extending the infrastructure to a distributed system is technically feasible.

We use the collected bounding rectangles as foundation for rendering selection highlights on top of application windows. For that purpose, the management application forwards the bounding rectangles to the window manager, which determines the appearance of the highlights and renders them on top of the desktop content. As a strong highlighting technique, we will present visual links across applications in Section 8.1.

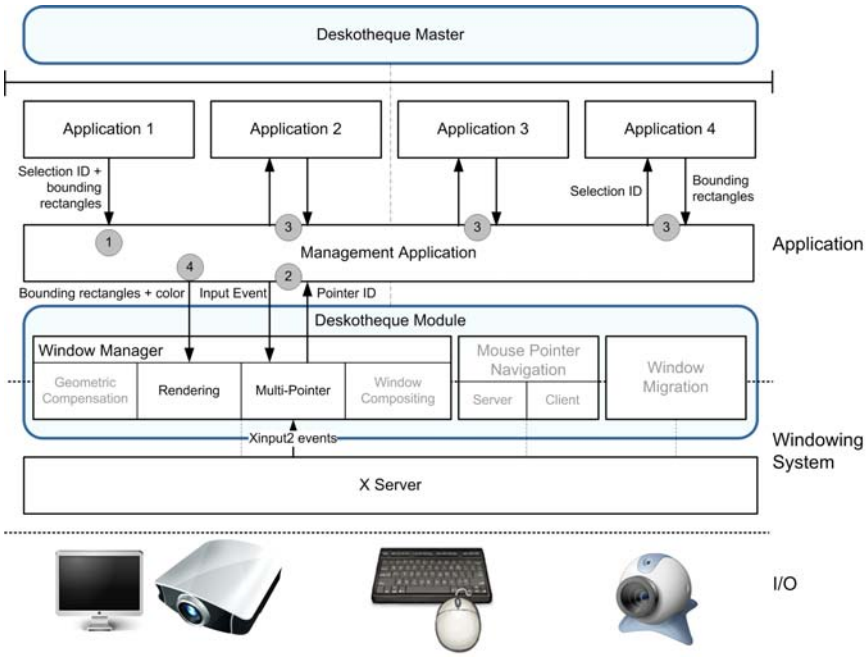


Figure 7.1: Central application coordination (on a single machine): (1) Application 1 reports a user selection and delivers the corresponding bounding rectangles of occurrences within the application. (2) The management application consults the multi-pointer extension of the window manager to determine the identity of the user triggering the selection and (3) forwards the selection string to all applications (accessible for this user). After all applications have reported the bounding rectangles of selection occurrences, (4) they are sent to the rendering plug-in of the window manager, together with the user's pointer color.

Application Modifications

An application utilizing the application coordination API needs to support three basic actions. First, the application has to *register* with the management application – either automatically at start-up or by user request – and report the visible window region to the manager. Second, it has to provide a user interface to *trigger source selections*, find other matches for the local selection ID, and deliver both, the selection ID and the associated bounding rectangles, to the manager. Third, it needs to be able to *process incoming selections* from the manager. Similarly to source selections, it has to find matching entities and send the bounding rectangles back to the manager. Additionally, a client application needs to report changes in the window content (*e.g.*, if the user scrolls the content) to the manager. If an application reports a content change, the manager checks for users currently having selections in the respective application, updates their selection rectangles, and sends the updated rectangles to the window manager.

Applications can be modified to facilitate the central application coordination API in three different manners:

Direct support: Software can be extended to directly utilize the interface provided

by the manager. As an example, we extended the *Caleydo* [142] multi-view framework. Whenever the user selects an element in one visualization view (*e.g.*, by hovering the mouse over a parallel coordinates poly-line), Caleydo reports the associated selection ID to the manager. Incoming selection requests are subject to Caleydo's internal ID mapping system to resolve ID relations in the biomedical domain.

Mashup: A web mashup combines functionality from an existing API with the management interface. We created a single HTML page utilizing JavaScript and the *Google Maps API*[*] for an interactive map application. When receiving a selection ID, the Google Maps API is queried for an associated geographic location. The location obtained from the first search hit is then converted to screen coordinates and a small bounding rectangle around this position is reported to the manager as selection region. The user can choose whether the map should be centered and zoomed to the retrieved geographic location or if the map should remain static. To trigger a geographical search, we provide a simple textual search box.

Software extensions: If existing software supports extension mechanisms (*e.g.*, via plug-ins), the required functionality can be implemented in a minimally invasive manner. We implemented an add-on for the popular cross-platform web browser *Mozilla Firefox*[†]. The add-on has full access to the DOM (document object model) of the displayed HTML-document. For each DOM element, the position in the browser window can be determined. This feature is utilized to find occurrences of the given selection ID string within the text passages of the document or labeled image elements. By temporarily enclosing the matching strings of text passages with an HTML-`<span>` tag, the position and size of the selection regions within the browser window can be retrieved. Communication with the management application to exchange selection IDs and selection region information is based on the `XMLHttpRequest` feature supported by the Firefox web browser. User selections are triggered by selecting text on the displayed website and pressing a button embedded in the browser UI.

The last category allows for usage of a wide range of applications with minimal effort, as many common web browsers and office applications provide extension interfaces. With these applications, a variety of use cases can be covered, because information from the web and common document formats are easily supported. However, when using such applications, data access is often restricted to textual content. As a consequence, ID mapping is limited to text parsing and string comparisons. In contrast, applications such as visualization frameworks often provide advanced interaction techniques for item selections and have ID mapping systems available – but often lack the extensibility required for a minimally invasive integration.

As applying the necessary extensions to existing applications is not always possible or feasible, we provide a light-weight version of synchronized highlighting for arbitrary, unmodified applications.

[*] http://code.google.com/apis/maps/
[†] http://www.mozilla.com/firefox/

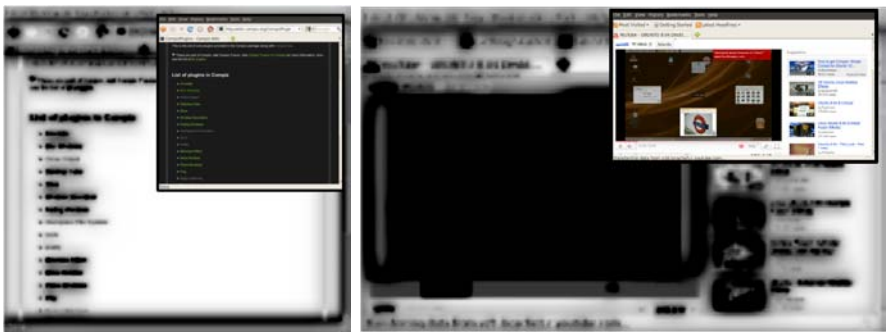
One-shot selections can be triggered from any unmodified application by selecting (marking) the desired text and subsequently pressing a keyboard shortcut. The management application then consults the operating system's selection buffer for the selection text and reports the selection string to all registered applications. However, using this approach, we can only determine the selection ID but neither the location of the selection region within the source window, nor additional occurrences of the selection ID in this application. Also, content changes in the unregistered source window are not propagated to the management application, so the location of the selection rectangle cannot be updated. Therefore, the user's selection ID will be re-set to the previous selection ID after a few seconds.

7.2.2 Window Importance Maps

Window importance maps are extracted from each window's content using a model of saliency-based visual attention. In a compositing window manager, the window content is represented as an image – more specifically as a texture, as we are relying on hardware-accelerated window rendering.

To determine the importance of each window's pixel, we rely on visual saliency. Saliency is a measure of how much a location visually stands out from the surrounding image regions [124, 156]. Typically used bottom-up saliency features are regions of high changes in luminance, color opposition, orientation changes, and motion. For our window importance maps, we apply the conspicuity analysis proposed by Mendez *et al.* [156], which extracts pixel-wise saliency values based on an analysis of lightness, red-green and blue-yellow color opposition, to the (dynamic) textures of the windows. In addition, we measure visual changes in windows and temporarily increase importance in regions where content has changed. Visual changes caused by user interaction, such as scrolling the window content, are ignored. The importance maps in Figure 7.2 show that user interface elements and information content, such as text or images, are assigned high importance values. In particular, the video in the window of Figure 7.2(b) is highly salient due to additional motion. Homogeneous background regions have low importance – independent of their background color.

Window importance maps are extracted from the individual windows' textures by using a GLSL implementation of Itti's visual attention model [124] by Mendez *et al.* [156]. The shader is applied each time the window is rendered by Compiz. Visual window changes in windows are monitored through window *damage* events, *i.e.*, notifications of window region updates provided by the X Window System. These damage regions are then amplified in the window's importance map.



(a) (b)

Figure 7.2: Examples for window importance maps: (a) a browser window showing primarily text and (b) a website containing a video. Importance gray values were inverted for illustration purposes. Insets show the unmodified window textures.

7.3 Display Adaptivity

Current window managers rely on a very simple spatial description of the available screen space: individual output devices are treated as rectangles, defined by their number of pixels. Spatial properties such as resolution, gaps between the displays, and orientation towards the user, are not considered. This unawareness is particularly problematic when using irregular projected displays: The desktop image is perspectively distorted and brightness is uneven when adjacent projections overlap, as visualized in Figure 7.3.

(a) (b)

Figure 7.3: A Linux desktop on an irregular display: (a) without any display adaptivity and (b) with geometric compensation and edge blending.

To effectively adapt the outcome imagery to the prevalent display factors, the window manager has do conduct two activities: First, adapting the window layout to the irregular display shape and second, correcting the resulting imagery for projection inhomogeneities.

7.3.1 Display Importance Maps

When using projectors, the resulting display outline is rarely rectangular. Oblique projection angles and combinations of multiple overlapping projections result in possibly concave

polyongal outlines. In addition, physical discontinuities, such as corners, further segment the available screen space.

In contrast to most related research on tiled projected displays, Deskotheque circumscribes the window manager's desktop rectangle around the 2D representation of the display (see Section 4.1 for more details). This leads to a loss of usable desktop pixels at the peripheral display areas, but in turn we can use all available projector pixels for information display and do not need to shrink the desktop imagery to fit within the largest inscribed rectangle. Instead, our idea is to explicitly exploit knowledge of the display form factor to adapt the spatial window layout, so all projector pixels are optimally used. This means that window placement in "invisible" desktop regions should be avoided. In addition, we refrain from placing windows across physical discontinuities.

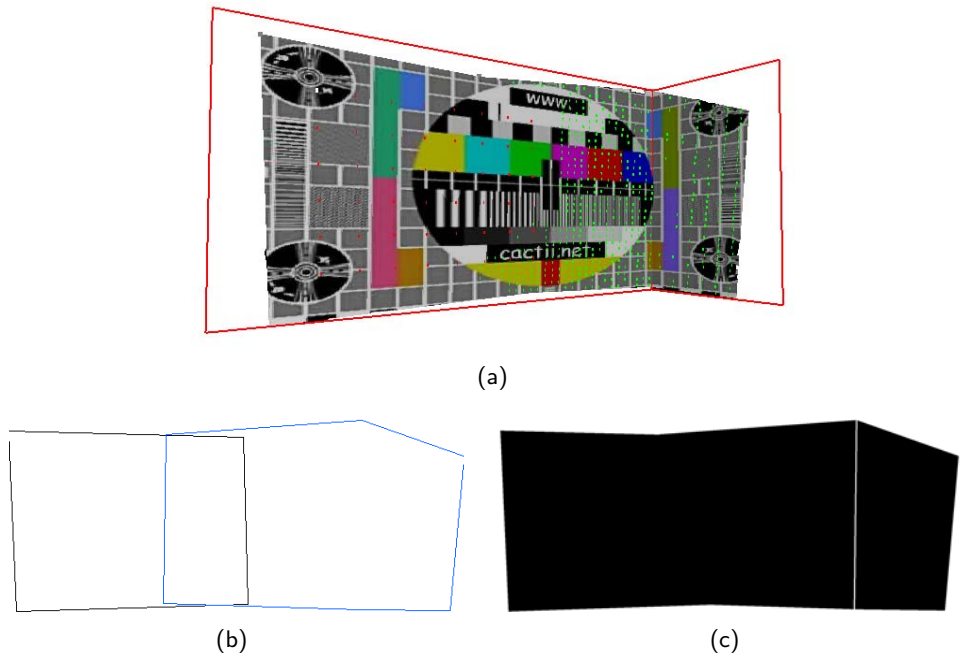


Figure 7.4: Display importance map of an irregular, tiled display (2 projectors) around a 90° corner: (a) the reconstructed model, where the textured quads represent the projection area and the red outline illustrates the circumscribed desktop rectangle. (b) From the display outline of the two projectors mapped in texture space, (c) the display importance map extracts usable (black) and unusable (white) pixels of the circumscribed desktop. Mind that the right projection spans two planes.

The description of usable and unusable pixels (*i.e.*, those cropped by the projection outline or along physical corners) is automatically derived from the spatial display model. For all co-planar polygons of a tiled projected display, the outlines of these polygons' unions are derived. The individual outlines are stored as a texture in screen size, which we refer to as *display importance map*. In this display importance map, usable pixels are black and the regions outside the union outlines are white, as shown in Figure 7.4.

The display importance map can furthermore encode user-defined regions, for instance if the user wants to leave a certain region of the desktop uncovered by any windows to allow for quick access to frequent desktop icons [113]. Alternatively, physical occlusions could be dynamically added as unusable display regions, like proposed in *occlusion-aware interfaces* [251]. Mind that such an approach would require constant capturing of the display, which is not available in Deskotheque's current implementation. Finally, the (estimated) user location could be seen as a focus region, where windows are more likely to be placed than in the peripheral areas [32]. To encode such focus and context regions, the usability (*i.e.*, encoded as gray value in the display importance map) of the display may gradually decrease with increasing distance from the closest user location.

The display importance map is automatically generated during the offline calibration process (Section 4.1). Alternatively, users can load manually created, static display importance maps at window manager start-up. Display importance maps are facilitated by importance-driven compositing (Section 7.4) to optimize the window layout for display-adaptive window management (Section 8.3).

7.3.2 Warping and Blending of the Desktop

Given a perfectly managed screen with all windows located within the visible display region, the window manager still has to ensure that the output image is undistorted and compensated for irregular brightness when using tiled displays. As input, the window manager receives the polygon outlines of all planar display regions for each output device, together with their associated homographies (*cf.*, Section 4.2). In the example of Figure 7.5, three polygons were delivered for the two projectors. In addition, the window manager receives one texture for each output device, encoding alpha values for edge blending of overlapping projection regions.

We extended the Compiz window manager to apply geometric compensation and blending by introducing an additional rendering pass. In the first pass, the X desktop is rendered into a frame buffer object. Then, the polygons are individually texture-mapped with this off-screen buffer content and warped with their associated homographies. For each output device, the alpha blending texture is applied full-screen on top of the warped polygons using an appropriate OpenGL blending function. Figure 7.5 shows the resulting output for a casually aligned, multi-planar multi-projector display.

To refine the warping result, the users can interactively apply simple geometric transformations to the desktop imagery – either by issuing keyboard shortcuts or by a simple GUI interface (Figure 3.4(a)). Translations, rotations, and scalings are applied to the screen's texture matrix.

As the mouse pointer is rendered directly in hardware by default, the cursor remains unaffected by desktop warping. This leads to an offset between the rendered desktop content and the actual cursor interaction space. This problem is circumvented by our multi-pointer extension of the window manager (Section 3.2 and Section 7.1), which ren-

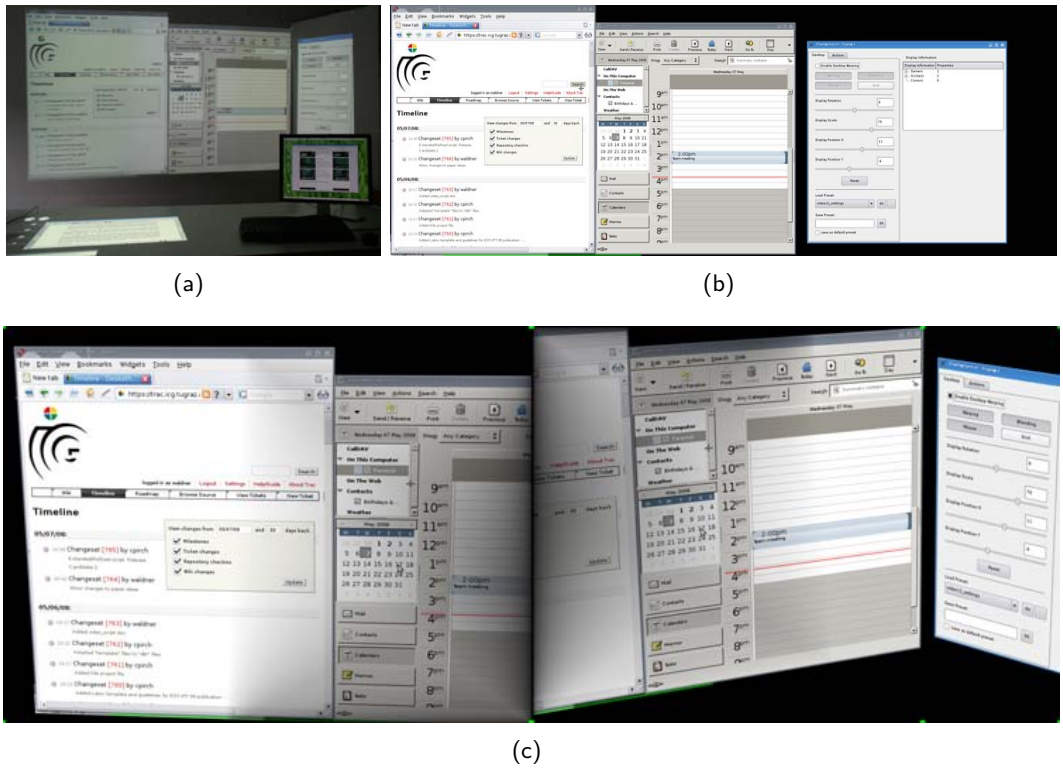


Figure 7.5: (a) For geometric compensation of the multi-planar, tiled display in the back, (b) the original desktop image is first rendered into an off-screen buffer. (c) Then, geometric compensation of the individual polygons and per-screen alpha blending is applied.

ders the cursor representations as textured and colored OpenGL quads, while switching off the X Window System's hardware cursor rendering. The mouse cursors thus contribute to the desktop content rendered into the off-screen buffer and are therefore subject to warping.

Mind that geometric compensation on the window manager level is only possible if the tiled display is driven by a single machine. If projectors are connected to different machines, a middleware has to take care that application windows can be shared across these machines and that a seamless navigation across the device boundaries is possible. Input redirection and window migration techniques can be used to accomplish this. The most challenging aspect, however, is the overlap region between adjacent projections: As shown in Figure 7.5(c), portions of the desktop need to be duplicated in these regions, in order to create a seamless image. However, duplicating windows to multiple machines on windowing system level is non-trivial, especially in combination with compositing window managers (*cf.*, background in Section 2.2.4). Thus, when building distributed tiled displays, using a remote desktop solution like VNC [196] (*cf.*, background in Section 2.2.2) is more appropriate.

7.4 Importance-Driven Compositing Window Management

Importance-driven compositing uses window and display importance maps to determine the importance of display and window regions. It uses this information for finding an optimal spatial window layout – in terms of visibility of important content – potentially in combination with see-through windows. Importance-driven compositing is the foundation for two window management techniques presented in the next chapter: uncovering windows (Section 8.2) and display-adaptive window management (Section 8.3).

Importance-driven compositing window management is composed of four basic steps:

1. The creation of window and display *importance maps* containing image-based descriptions of important regions (sections 7.2.2 and 7.3.1), which are accumulated into a common desktop importance map,

2. a *window layout* routine, placing windows to minimize the overlap of important content,

3. importance-driven see-through *compositing*, applying per-pixel transparencies to reveal important content of occluded windows, and

4. *interaction* techniques allowing the user to access and manipulate content in occluded windows.

The optimal window layout can be determined for both, opaque windows (*cf.*, Section 8.3 and Figure 7.6(a)) and semi-transparent windows using importance-driven see-through compositing (*cf.*, Section 8.2 and Figure 7.6(c)). When employing importance-driven see-through compositing without optimizing the spatial window layout, content-aware transparencies are applied to unimportant regions in occluder windows (Figure 7.6(b)), similar to *free-space transparency* proposed by Ishak and Feiner [123]. If covering all four steps, the algorithm aims to find the spatial window layout so the maximum amount of important information can be displayed by facilitating pixel-wise transparencies, as illustrated in Figure 7.6(c).

(a) (b) (c)

Figure 7.6: Importance-driven compositing facilities: (a) optimized spatial layout for opaque windows, (b) see-through compositing to reveal occluded content, and (c) the combination: optimized spatial layout for see-through windows.

Importance-driven compositing is implemented as plug-in for the Compiz window manager and facilitates advanced GPU languages – namely, the OpenGL Shading Language[‡] (GLSL) and the Open Computing Language[§] (OpenCL) – to support real-time interaction. In the following, the above mentioned steps will be discussed in more detail.

7.4.1 Desktop Importance Map

We use importance maps as unified image-based representation of importance in windowing systems. Display importance maps (Section 7.3.1) and window importance maps (Section 7.2.2) are accumulated into a common desktop importance map, describing the distribution of information on the entire display. In Figure 7.7, the desktop importance map (0) is a combination of a single display importance map (1) and multiple window importance maps (2) by the overlapping windows on the screen.

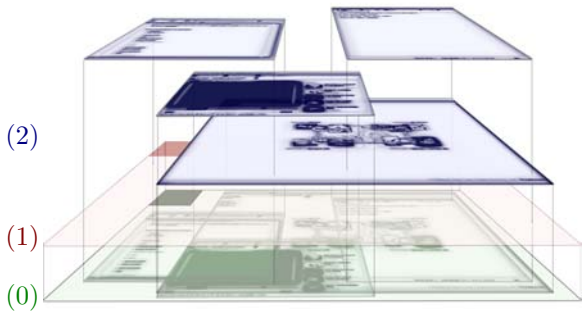


Figure 7.7: Window importance maps (2, blue) are created for each window individually and accumulated with a display importance map (1, red) to a common desktop importance map (0, green). Dark areas represent high importance.

7.4.2 Window Layout

The aim of the window layout routine is to spatially arrange windows so that important regions of occluded windows are uncovered, if possible. Figure 7.8 shows the layout step for a single window and the resulting desktop composition. The optimal window placement is determined by considering three influencing factors described in the following (*cf.*, Figure 7.8(4-6)).

Firstly, the information overlap of the window with the already existing information on the desktop (I_d) shall be kept low. For this information overlap measure, we distinguish two cases. In the case of opaque windows, the information overlap describes the amount

[‡]`http://www.opengl.org/documentation/glsl/`
[§]`http://www.khronos.org/opencl/`

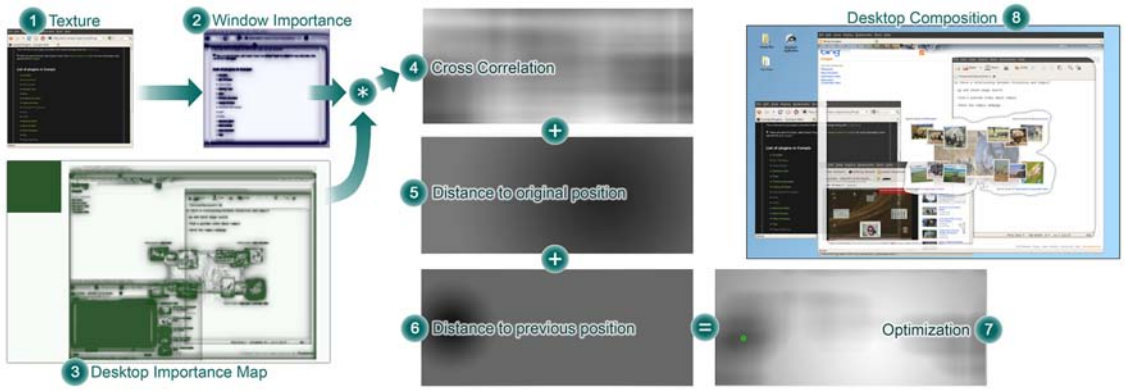


Figure 7.8: Overview of importance-based compositing for a single window: (1) from the window's texture, (2) the window importance map is created. (3) Given the desktop importance map, the window layout routine finds the optimal placement considering (4) the information overlap, (5) the distance to the original window location, and (6) the distance to the previous location. (7) The combination of these factors gives an optimization problem with the solution marked as a green dot (window center). (8) Finally, the window is rendered to the existing desktop content.

of information on the desktop, covered by the window rectangle:

$$J_i(\boldsymbol{p}) = \sum_{(x,y)\in\Omega_w} I_d\left(\boldsymbol{p} + (x,y)\right), \tag{7.1}$$

where (x, y) visit all discrete positions in the window rectangle Ω_w, and $\boldsymbol{p}$ is the window location. In the case of windows being rendered with see-through compositing, we additionally need to consider the information contained in the window (I_w). The information overlap can be formulated as a cross-correlation of the two importance maps I_w and I_d, as illustrated in Figure 7.8(4):

$$J_i(\boldsymbol{p}) = \sum_{(x,y)\in\Omega_w} I_w(x,y) \cdot I_d\left(\boldsymbol{p} + (x,y)\right). \tag{7.2}$$

Secondly, the location $\boldsymbol{p}$ should have little displacement from the original window location $\boldsymbol{p}_o$, which is defined by the location where the window has been mapped or manually positioned by the user (Figure 7.8(5)). Thus, this term is responsible for maintaining a certain degree of spatial stability.

Thirdly, the resulting window location $\boldsymbol{p}$ should vary minimally from the location in the previous frame $\boldsymbol{p}_p$ (Figure 7.8(6)). In other words, jitter should be minimized.

These requirements can be formulated as an optimization problem over all possible window locations $\boldsymbol{p}$ (Figure 7.8(7)):

$$J(\boldsymbol{p}) = \omega_i J_i(\boldsymbol{p}) + \omega_d D(\boldsymbol{p}_o, \boldsymbol{p}) + \omega_j D(\boldsymbol{p}_p, \boldsymbol{p}), \tag{7.3}$$

where J is the associated cost function to be minimized, composed by a weighted

sum of the information overlap J_i, and the distance D to the original location (window displacement) and previous position (jitter). The individual weights (ω) vary for the emerging window manager techniques, such as presented in Sections 8.2 and 8.3.

To reduce the number of potential window locations ($\boldsymbol{p}$), the search space can be decreased by additional constraints. For instance, windows can be bound to certain screen regions or to "parent" windows. In addition, we limit the maximal window movement over time. This introduces smooth frame-to-frame animations and helps the user keep track of window movements. Window content is never placed outside the screen boundaries. This decreases the search space for large windows, while maximized windows will not be re-positioned at all.

Our algorithm treats multiple windows sequentially in a greedy manner. For each window, the best placement is determined by solving the optimization problem as stated above. Subsequently, the window's importance map is added to the desktop importance map at the determined location. The modified desktop importance map then serves as an input for the next window's placement. Thus, windows being traversed first have more freedom in finding a good placement. For our window management techniques (*i.e.*, uncovering windows in Section 8.2 and semi-automatic window coordination for display-adaptive window management in Section 8.3.2), the inverse window stacking order – defined by the windows' recency of use, beginning with the window currently holding the input focus – was used as traversal order. While this traversal order is appropriate for work on small-scale displays, we found evidence that on large-scale displays different traversal sequences might be more appropriate, such as defined by the proximity to the currently active window (*cf.*, Section 9.4).

The evaluation of the window layout cost function is a computationally expensive operation and is therefore implemented in OpenCL on the GPU. For each window being rendered by Compiz, the OpenCL layout routine returns the optimal window translation, which is then applied as transformation to the window's quad, or is issued as window move command to the X Window System via Xlib.

7.4.3 Compositing

The compositing step is responsible to correctly render the windows to the screen. In contrast to conventional window management, we render the windows according to their priority for the layout algorithm, which usually does not correspond to the conventional rendering order – the stacking order defined by the recency of use. In the current implementation, we render windows front to back to ensure higher priorities for recently used windows.

In the simplest case of opaque windows, the compositing step renders windows as if traversed from back to front, as usual, by setting the current window pixel's alpha value

$\alpha_w(x, y)$ as:

$$\alpha_w(x, y) = \begin{cases} 0, & \text{if } \alpha_d(x, y) > 0.0 \\ 1, & \text{else,} \end{cases}$$

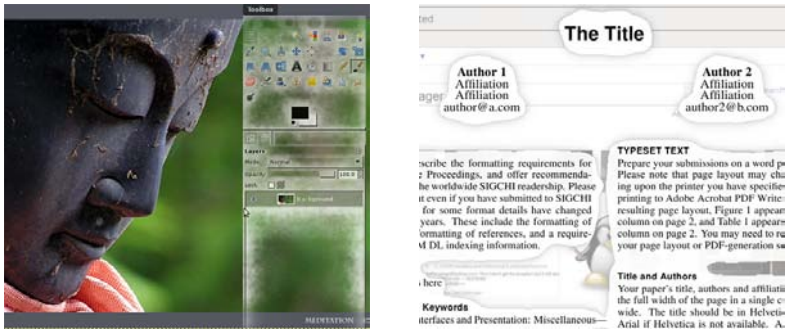
where $\alpha_d(x, y) > 0.0$ indicates whether the desktop pixel at (x, y) is already occupied by a higher level window.

For increasing the amount of visible information on the screen, importance-driven see-through compositing reveals occluded content by applying pixel-wise transparencies. We implemented two well-known compositing techniques from the field of technical illustrations and volume rendering [250]: *ghosting* and *cut-aways*.

Ghosting determines each window pixel's alpha value $\alpha_w(x, y)$ by evaluating the importance ratio of the window's importance map $I_w(x, y)$ and the desktop importance map $I_d(x, y)$ at the respective pixel:

$$\alpha_w(x, y) = \frac{I_w(x, y)}{I_w(x, y) + I_d(x, y)}.$$

Thereby, it ensures that the most important features of each window are visually preserved (Figure 7.9(a)). However, if the window layout routine cannot spatially separate important regions (for instance due to high information density), important features in overlapping windows compete for visual prominence.



(a) (b)

Figure 7.9: See-through compositing: (a) Ghosting and (b) cut-aways.

Cut-aways put more emphasis on the windows' stacking order: they ensure that the most prominent features of the overlay windows are preserved. Only if the desktop's importance map value is below a certain threshold, the obscured window's content is revealed. Smooth blurring and desaturation of obscured window content provide subtle depth cues, such as a slight blurring and de-saturation. This approach is similar to free-space transparency [123], where transparency is only applied to white regions in overlay windows. Our cut-away technique differs from free-space transparency, as we apply hard boundaries between the foreground and the background and add shadows to visually

indicate depth layers (Figure 7.9(b)).

The compositing step is implemented as GLSL fragment shader. It determines per-pixel alpha values and evaluates each pixel's neighborhood for blurring and shadowing, according to the chosen compositing technique.

7.4.4 Interaction

Importance-driven compositing window management allows users to interact with visible portions of occluded windows – even if located within the boundaries of an overlay window. We rely on the simple assumption that the user aims to interact with the visually most prominent window at the current mouse pointer location. We therefore set the input focus according to the compositing result: the window with the overall highest contribution to the pixel's color below the mouse pointer is activated.

The most salient window to receive the input focus at the current cursor location is determined in the GLSL compositing shader. The information is queried each time the mouse was moved. To redirect the mouse input to the currently active window, we raise the active window in the window manager's stacking order, so input is reliably forwarded to the respective window, even if located underneath a top-level window. Although modifications to the window manager's stacking order usually lead to a change of the rendering order, we do not alter our traversal order for layout and compositing to keep the desktop visually consistent.

Depending on the employed window management technique, window management using importance-driven compositing introduces a noticeable latency. On a Quad-Core 2.80 Ghz CPU and NVIDIA GeForce GTX 480, for a desktop resolution of 1280x1024, placing and rendering a window of approximately 500x300 pixels requires 6ms. For a conventional office scenario with five managed windows, we obtain an average frame rate of 20 fps. However, as importance-driven compositing is usually only temporarily activated, these frame rates are acceptable.

7.5 Discussion

In this chapter, more or less fundamental changes to conventional window management have been described, which will act as foundation for the window management techniques presented in the next chapter. For operation in emerging display environments, combinations of these extensions will be facilitated by certain techniques. For instance, visual links for multiple collaborators on a tiled display (Section 8.1) will facilitate the multi-user extensions, warping and blending, as well as the central application coordination routine. Display-adaptive window management (Section 8.3) uses warping in combination with importance-driven compositing to optimize window placement according to the display importance maps of irregular displays.

However, the techniques in the following chapter only partially exploit the possibilities

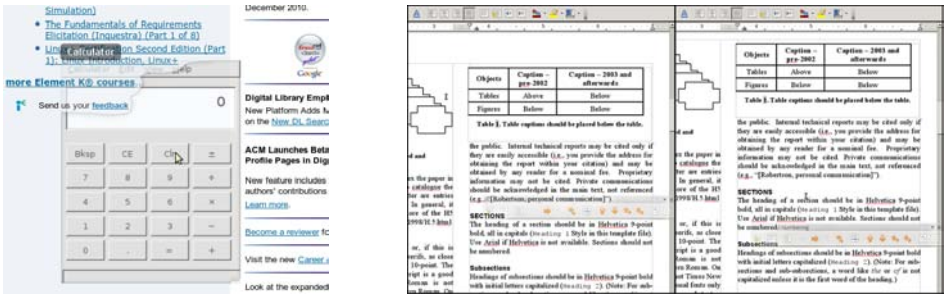
provided by these extensions. Using some of our extensions, previously presented window management techniques could have been implemented more efficiently, as a few of the following examples will illustrate. The central application coordination mechanism could be used to improve recognizability of automatically down-scaled windows in tiled window managers, as proposed by Miah and Alty [157]. In their concept, important window elements remain at their original size or get highlighted. With our synchronized highlighting mechanism, important elements can be determined by the user selection in a focus window, while less important window elements are automatically shrunk by the window manager. Similarly, important window elements may be automatically cut out of the surrounding window, creating free-floating window *cuts* [238], *snips* [114], or *clips* [155]. In contrast, previous work has relied on the user to manually cut out important regions [114, 155, 238]. As a semi-automatic approach to window region detection, the user may paint a stroke on top of the anticipated window region. An image-based segmentation algorithm then extracts the associated context region by analyzing the window importance maps (Figure 7.10).

Figure 7.10: Conceptual sketch of semi-automatic region segmentation: on an application window, the user paints a rough stroke to indicate the region of interest. The segmentation is conducted on the window importance map. The duplicated window region can be treated as individual application window, as proposed by Tan *et al.* [238].

Importance-driven compositing can also be used as alternative approach to user interface *holes*, as presented by Stürzlinger *et al.* [232]. User interface holes are used to reveal auxiliary GUI elements of windows underneath. With importance-driven compositing, auxiliary applications (*e.g.*, small widgets) or floating toolbars can be positioned relative to a parent window and be made visible using see-through compositing. Figure 7.11 shows two screenshots of floating windows acting as user interface holes.

Of course, our window manager extensions provide room for improvements. The central application coordination mechanism relies on very simple identity relationships to synchronize user selections across applications. Indeed, applications themselves can apply some specialized ID mapping – as demonstrated in the extension of the *Caleydo* [142] framework – but for applications extended in a minimally invasive manner, such as our web browser add-on, this is not easily possible and definitely not feasible. Sophisticated ID mapping therefore has to be taken care in the central management application itself. It should be possible to load special task-related ontologies to resolve more complex relations in a certain task domain.

Importance-driven compositing relies purely on image-based analysis of bottom-up



(a) (b)

Figure 7.11: Floating (a) auxiliary applications and (b) toolbars are positioned automatically within a main window with minimal information overlap. Mind how the floating toolbar in (b) is re-positioned as the document is scrolled.

saliency features of window textures to derive "important" window regions. Although this measure is more sophisticated than simply defining white pixels as unimportant and the rest as important (as, for instance, in [123]), in many cases the window regions attracting our visual attention are not necessarily the regions that are most important for an information analysis task. As an example, an advertisment in a web browser window may be much more salient (and therefore more important to our algorithm) than the actual information the user is looking for – which may be "just" text. As another example, menu bars in windows are highly salient but are not very important in secondary windows, where the user is merely looking at the content but is not actively interacting. Stürzlinger *et al.* [232] discussed how user interface elements, like menus or buttons, could be automatically detected within the window manager by facilitating accessibility APIs. Incorporating such a high-level knowledge into the concept of importance-driven compositing may improve usability significantly. Also, increasing the importance of selection regions, collected by the central application coordination instance, in the window importance maps may help to decrease the importance of semantically unimportant, but visually salient window elements.

Chapter 8

Window Management Techniques

In the previous chapter, extensions to the basic window manager functionality for operation in emerging display environments have been presented. It has been shown how window managers can be extended to incorporate knowledge of the environment (*i.e.*, display form factors and users) and the window layer.

In this chapter, this functionality will be facilitated to create new window management techniques to support users in discovering, accessing, managing, and sharing information in emerging display environments. Table 8.1 shows an overview of the window management techniques. The comparison lines out whether the individual techniques (or a special mode of the techniques) are particularly suitable for certain display or user configurations.

	Visual links	**Uncovering windows**	**Display-adaptive window management**	**Polarization-based interfaces**
Primary purpose	information discovery, information filtering, information sharing	information discovery, information access	window management, information discovery	information sharing, information comparison, information filtering
Levels	W, E	W, S	W, S, E	(S), E
Small displays	–	x	–	–
Large displays	x	–	x	x
Irregular displays	–	–	x	–
Multi-display	x	–	–	–
Multi-user	x	–	–	x
Sections	8.1	8.2	8.3	8.4

Table 8.1: Comparison of window management techniques: the primary purpose(s) of the technique, the level from which the technique derives knowledge from (W = window, S = screen, E = environment), whether it was specifically designed for different display or user configurations (x: yes, –: no), and the respective sections in this chapter.

8.1 Visual Links

With increasing display space, users tend to have a larger amount of open application windows [112]. With the increased amount of space, they can arrange multiple application windows containing diverse information next to each other with no or only very

little overlap. As a result, a large amount of information from different sources can be visualized simultaneously, which supports the user in a complex information analysis task by decreasing the amount of explicit window switching operations.

However, visually locating items of interest on a large display scattered with information is one of the greatest challenges when working with very large displays [234]. Items of interest for a particular task may be distributed to multiple application windows and available in diverse visual represenations – like text, graphs, images, or maps. Information may be partially outside the user's field of view – in peripheral display regions or on discontinuous display locations. Guiding the attention to peripheral items or showing relationships between information across application windows thus requires strong visual cues to guide the user's attention to locations of interest and to show relationships and patterns explicitly.

Visual links across applications facilitate our centralized approach to synchronized highlighting in the window manager (Section 7.2.1) and visualize related entities contained in multiple application windows using strong visual cues: connection lines.

8.1.1 Visual Links Highlighting Technique

Supportive information located at the periphery of the display is likely to be overlooked, even when visually highlighted. Typical highlighting techniques are rendering the items of interest in distinct colors or surrounded by frames. Other highlighting techniques draw the user's attention to items of interest by reducing the visibility of the surrounding information. Common methods are to decrease saturation [270], brightness [138, 270], or sharpness [139]. All of these cue-based focus and context techniques [59] either apply image-based modifications to the background image or render additional information on top of the output imagery. Thus, they can be easily applied in a compositing window manager, where full access to the individual window textures and the final desktop composition, respectively, is available.

In an experiment by Hoffmann *et al.* [108], users committed more errors and were more annoyed with a technique darkening the context regions, compared to techniques highlighting the focus object with frames or trails. The also found that for targets appearing at a large distance from the focus window, trails to the target location performed better than highlighting the target with a colored frame. They identified curved, asymmetric trails to be more easily detectable, as they are more distinguishable from the merely rectangular screen content.

Routing

Following the guidelines by Hoffmann *et al.* [108], we indicate related regions by rendering frames around the selection regions, as well as by connection lines from the user's current interaction window to the selection regions. If multiple selection regions are reported

in a single application window, connection lines are bundled [110] to reduce visual clutter. Bundling also clearly indicates relatedness of selection regions with regard to their application window.

We distinguish two window types: The **source window** is the application window where the user selection has been registered. As the user focus is currently on this window, all visual links to related information emerge from this spot. If multiple selection regions are located in the source window, each region is highlighted and connected to a bundling point in the center of gravity of all regions (*cf.*, Figure 8.1 A), where the connections to the target windows emerge.

Target windows are all registered application windows where no user interaction is taking place and at least one selection region was reported. The main purpose of visual links is to lead the user's focus to these windows and to express the relationship of relevant items with respect to the source window. Each target window reporting selection regions is linked with the source window by a single connection line. This connection line bundles the connections to all selection regions in the target window. In contrast to the source window, the bundling point is set to the intersection point of the line connecting source and target window with the target window's boundary. From this bundling point, the individual connection lines to all selection regions emerge (see Figure 8.1 C-D).

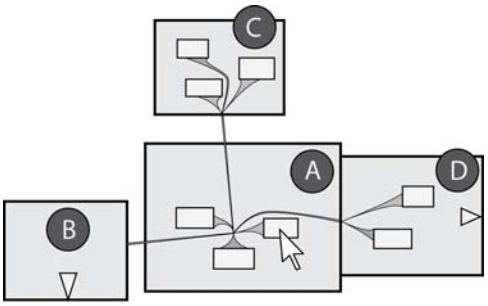


Figure 8.1: Visual links emerging from a source window (A) to three target windows (B-D). Target windows B and D contain hidden items, indicated by an arrow. Mind the connection lines in window A and C avoiding selection region obstacles.

To avoid occlusion of potentially valuable information, rendering connection lines across selection regions is avoided. If a connection line would intersect a selection region, the corner point of the intersected region leading to the shortest non-intersecting path is added to the connection line. This results in the connection being "curved" around the region (Figure 8.1 A and C).

Off-Screen Visualization

If an application reports selection regions outside the visible window region, arrows at the window boundaries give a visual cue about invisible related information (see Figure 8.1 B and D, as well as Figure 8.3(c)). Contrary to some off-screen visualization techniques

encoding the exact location of off-screen items, like *Halos* [23] or *Wedges* [92], we group invisible regions into four off-window directions (up / down / left / right), according to the four scroll directions of textual documents. The width of the arrows is static, while the length encodes the number of hidden selection regions in this direction, giving the user a cue about the amount of invisible information in each direction. If a target window has no visible, but only hidden selection regions, visual links are drawn only to the window border (*cf.*, Figure 8.1 B).

Rendering

Connection lines are rendered as Bézier surfaces, emerging from the window's bundling point with a given line width and expanding to the selection region border up to a given maximum width. Links are rendered by the window manager on top of the existing desktop content, as shown in Figure 8.2. In its current implementation, we do not incorporate knowledge from the screen layer (*i.e.*, window arrangements) and therefore do not consider (partially) occluded application windows. Thus, highlight regions for occluded items are always rendered on top of the top-level window. In the future, we plan to combine visual links with importance-driven compositing (Section 7.4) to guarantee visibility of otherwise occluded selection regions using see-through compositing.

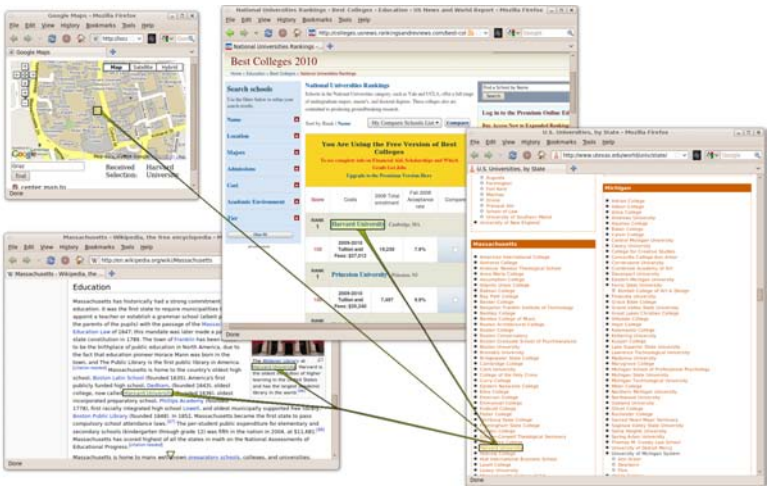
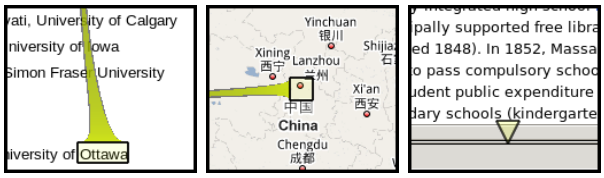


Figure 8.2: Using visual links to relate textual information on web pages with a geographic location: looking up Harvard University.

Connections are rendered half-transparently, so underlying data can still be identified. "Shadows" surrounding the connections help to discriminate visual links from the desktop content. Figure 8.3(a) and (b) shows highlight regions and their associated connection line for highlighting text and a map location.



(a) (b) (c)

Figure 8.3: Close-ups on single connection lines and selection regions highlighting (a) text and (b) a map location. Hidden items on a website are indicated by an arrow (c).

User Interaction

A new set of visual links is created each time the user selects an item in a registered application window. Connection lines, selection region highlights, and arrows are displayed until a new selection by the owner, a change in window size or location, or a change in window content (*e.g.*, by scrolling) is registered. Links are rendered in high opacity after the user has initiated a selection but fade to a low alpha value after a few seconds. The rectangular highlight regions around the reported selection occurrences remain at full opacity. In this way, readability of content is improved. By moving the mouse over one of her selection highlights, the user can set back the links to their initial alpha value. For fading the links to a lower alpha level manually or switching them off entirely, users can employ simple keyboard shortcuts.

8.1.2 Collaborative Information Linking

Information analysis is often a collaborative task. Examples range from families planning a holiday trip to medical experts discussing the treatments of their patients. What these examples have in common is the need to extract relevant information from multiple sources – from web sites and maps to patient records and biomedical visualizations – and to share these pieces of information with the collaborators to find consensus. With the raise of large-scale displays and multi-input support, these collaborators can facilitate a shared platform for co-located collaborative information work.

However, sufficient display space to arrange personalized information and concurrent input support alone does not ensure a successful collaboration. Common problems of co-located collaborative information management – such as the inability to follow other users' cursors [264], problems resolving deictic references [101, 119, 204], and distractions caused by other users' cursor movements [258] or changes to the spatial display layout [72] – cannot be easily resolved in such *collaboration-transparent environments* [140]. Basic *mechanics of collaboration* [95, 179] for smooth teamwork need to be supported by the system.

Collaborative information linking extends the concept of visual links across applications to multiple concurrently operating collaborators. Each user in the environment is uniquely

associated with a pointing device and a distinct color, also used for cursor rendering (Section 4.3). As the user selects an item in an application window, the management application retrieves the user's identity from the window manager, retrieves the associated selection rectangles from all other applications, and sends the collected rectangles to the window manager for routing and rendering (*cf.*, Section 7.2.1). Visual links are rendered in the user's associated color and emerge from the window for which the user selection has been reported. Figure 8.4 shows an example workspace operated by two users conducting a biomedical analysis.

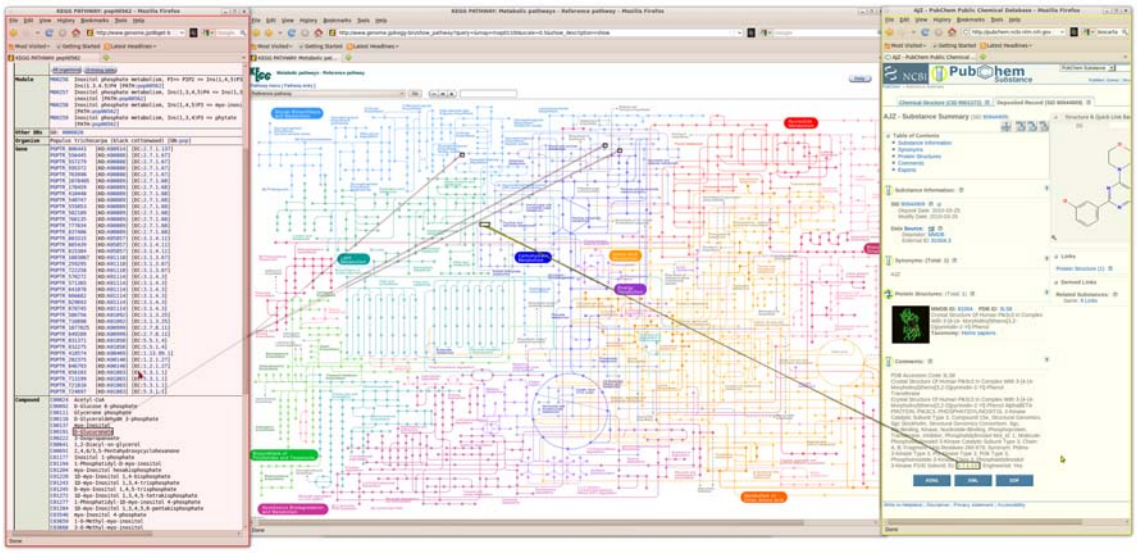


Figure 8.4: Screenshot of a shared workspace: color-coded sets of visual links connect text selected in two protected browser windows with matching occurrences in a shared overview chart.

Protected and Shared Workspaces

Visual links by collaborators may potentially lead to interference for other users, as links are automatically created upon a user's selection. As a result, newly created sets of visual links may unintentionally intrude into other users' personal territories and interrupt their current work. In contrast, accessing another user's workspace with the mouse pointer (input conflicts) is a conscious activity, and can therefore be mediated socially much more easily.

We therefore provide the possibility to protect workspace areas from other users' links. These areas may be dedicated private displays or single application windows on shared displays. The user can protect any window in the workspace, which is not already locked by another user, by moving the mouse cursor to the desired window and pressing a shortcut. With the same procedure, a protected window is released. Protected windows are visualized by color-coded boundaries in the users' assigned colors (see lateral browser win-

dows in Figure 8.4). Private displays are locked for other users' links by default. The user can release this lock in a GUI.

Protection is coordinated in the window manager. Window (un)lock events are propagated to the management application, which controls access restrictions between users and applications. If a window is locked by a user or located on a user's private display, incoming selection IDs by another collaborator are not forwarded to this particular application associated with the window.

Selection Hijacking

Users can test whether others' selections occur in their private workspace by temporarily releasing window protection or by "hijacking" a specific collaborator's selection. To hijack a selection, the user moves the cursor over one of the collaborator's selection rectangles on a shared window. The user's links are then temporarily replaced with the collaborator's selection, while the previous highlight regions remain visible to explicitly show the relationship between the two selections (Figure 8.5). Hijacking is active as long as the user keeps the cursor within the selection region, with a minimum display time of one second.

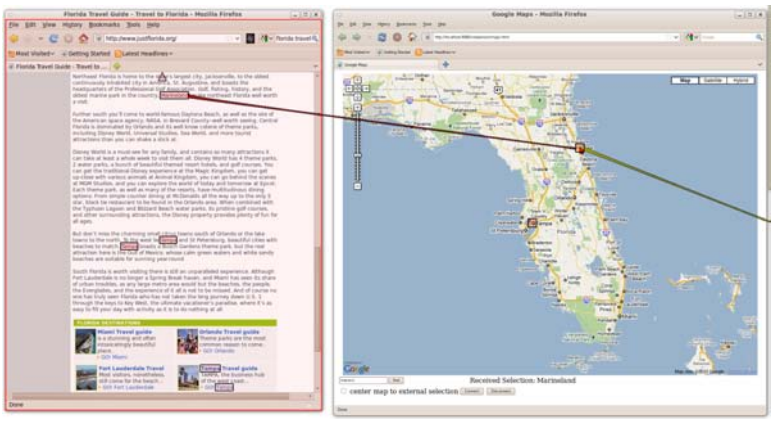


Figure 8.5: Selection hijacking: the red user moved the cursor on top of the other user's (yellow) selection on a shared window. The connection lines for the current selection "Tampa" are substituted with the yellow user's selection "Marineland", but the highlight regions of "Tampa" remain for easier comparison.

Selection Storage and Management

We provide a "bookmark list" as a central storage tool, where users can store their current selections for later investigation. In addition, users can employ it as a global search tool by entering arbitrary selection strings and testing whether this selection is available in any registered application window. The bookmark list displays each user's current selection string together with a bookmark button. By pressing this button, a new bookmark appears as button labeled with the selected text (Figure 8.6). By pressing such a bookmark button,

links to the currently visible application content are created. In this way, users can quickly store findings as selection strings and test them later on new window content.

Figure 8.6: The yellow user selected a bookmarked item ("Portugal") from the bookmark list.

This is a different bookmarking concept than provided by conventional web browsers, which store an application state or content rather than a user's search string. Our implemented selection bookmarking does not allow users to restore the applications' states at the time of bookmarking their selection. To provide such an environment-wide bookmarking, registered applications additionally would have to provide additional context information with every user selection report – such as their current URL, document, or visualization view – and the possibility to restore these states upon bookmark selection. Alternatively, a system-wide time-centric information management approach (similar to [194]) could restore the entire system's previous (visual) state by storing screen-shots of the window content at the time of bookmarking. However, restoring a previous system state can lead to interference for collaborators. Thus, privacy mechanisms – such as restricting reconstitution to unprotected application windows – would be required. In addition, the visual presentation of the windows' previous states has to be well-designed, in particular if the window layout has changed in meantime.

8.1.3 Cross-Display Information Linking

Collaborative information linking is particularly interesting for multi-display environments. As a motivating scenario, consider the collaborative information visualization scenario by Streit *et al.* [229] (Figure 8.7): Multiple expert users from different domains (*e.g.*, histopathologists, oncologists, surgeons) are undertaking a collaborative analysis of a patient's gene data, histological data, and data obtained from external databases. Information is scattered across multiple displays within the environment – partially outside the user's field of view. To discover these peripheral pieces of information, visual links across display boundaries can lead the user's gaze even to very distant locations of the environment. They indicate the existence of relevant information despite physical occlusion,

outside the user's field of view, and even though the information is too distant and thus too small to be perceived by the user.

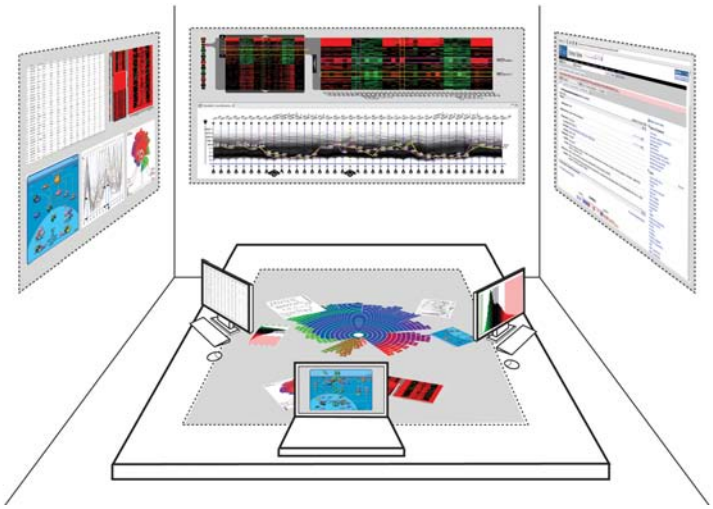


Figure 8.7: Mockup of a shared multi-display workspace for diagnosis in a clinical setting [229].

Visual links connecting items located on distinct displays do not only need to bridge display space potentially covered by other applications. They also need to bridge display-less space between discontinuous displays. Cross-display links therefore require a spatial description of the environment to calculate which displays have to be crossed and at which display locations the start and end point of the visual link fragment has to be set. We provide two multi-display coordinate systems for rendering cross-display visual links: pair-wise planar mappings of the environment and a perspective representation of the environment, as perceived from the owner's estimated field of view (Section 4.4).

Using pair-wise planar mappings, links to adjacent displays are always crossing through a spatially limited interval on a particular display edge – or even through a fixed entry and exit point, as illustrated in Figure 4.17. Links using this mapping are highly predictable and work equally well from every perspective.

Using a perspective mapping, links are routed as seen from the estimated owner's perspective, as shown in Figure 8.8. The resulting links seem less complex from the user's point of view, as the link direction is more consistent across the individual displays. Perspectively mapped links and links using a pair-wise planar mapping have not been compared in a controlled experiment. However, informal user feedback from the setup in Figure 8.8 indicates that perspective links are much easier to understand and follow. Mind that perspectively mapped links with a non-tracked user have a limited application area. In the example mock-up of Figure 8.7, displays located behind the user cannot be included in the user's perspective map and are therefore unreachable for cross-display links.

(a) (b)

Figure 8.8: Cross-display links using a perspective mapping, as seen from the left (a) and right (b) user's perspective. Even though the link colors are similar (yellow and green), the links can be discriminated based on their origin direction on the respective home display.

8.2 Uncovering Windows

Information analysis tasks mostly require multiple sources of information to complete a single task. In addition, users are often involved in multiple tasks at the same time, requiring a frequent switch between their active information sets – or, more particularly, application window sets. Floating menus and multiple windows per application additionally contribute to the large amount of windows cluttering the standard desktop screen. Even if the display size increases, users just tend to keep more application windows open [112], leading to overlaps between windows and little uncovered desktop space.

As a consequence, researchers [87, 112] found that window switching (*i.e.*, bringing occluded windows to the front) is a frequent activity in today's window managers. Hutchings *et al.* [112] also showed that most windows are activated only for a very short time period – typically less than four seconds. This indicates that users often only need to skim the content of a secondary window to resume their primary task, such as reading a bit of documentation, checking if a new e-mail has arrived, or retrieving the result of a calculation. With sequential window switching techniques, the user has to perform multiple operations: initiate the window switch, identify and select the window of interest, perform the actual operation, and repeat the window switching step to return to the previous window. Not surprisingly, "the need to Alt-Tab" has been described as tedious [91].

To reduce the number of necessary activities to quickly access occluded content, we use importance-driven compositing (Section 7.4) to uncover information in occluded windows on demand. By pressing a keyboard combination, automatic spatial window layout and cut-away compositing is applied to all (partially) occluded desktop windows, which are currently not minimized (*cf.*, Figure 8.9).

We apply a low penalty on window displacement (ω_d in Equation 7.3) on the layout algorithm to give more emphasis on minimizing the information overlap than a stabilized window layout. Thus, on a large display or a display containing a few number of windows

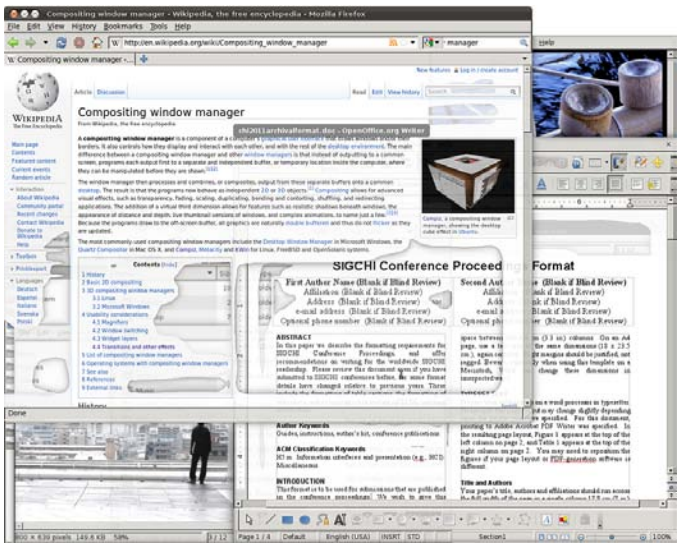


Figure 8.9: Uncovering windows on an SXGA display: unimportant window regions are temporarily cut away and occluded windows are spatially arranged to reveal obscured content. The user directly interacts with the otherwise invisible document content uncovered below the browser window.

(as shown in Figure 8.10), occluded windows are rather moved towards empty desktop regions than being placed below unimportant window regions of overlay windows.

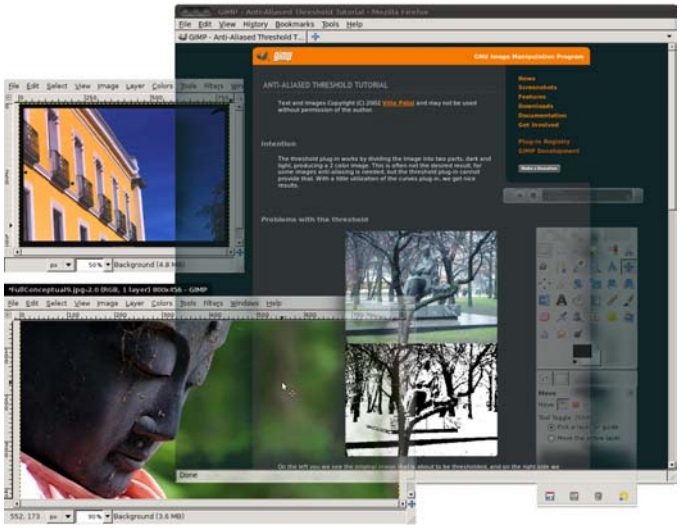


Figure 8.10: Uncovering windows with a small number of windows: windows are rather moved to empty desktop regions and are only rendered within occluder window boundaries, if the information overlap cannot be further minimized.

We discard the window title bars from occluded windows, as they have a high visual saliency but little benefit for user interaction in this situation. In addition, we use subtle *dynamic transparency* [93], which increases an occluded window's overall alpha value if the

pointer moves closer or if the window currently holds the input focus. As input is redirected to the most salient window at the cursor location, simple operations in occluded windows – such as pressing a button – can be accomplished while exposing occluded content without actually bringing the window to the front. To signal which window currently holds the input focus, we increase the active window's overall alpha value and show its title at the left upper window corner (as for the document editor in Figure 8.9). When releasing the key combination, the active window under the pointer is brought to the front and the other occluded windows return to their original locations.

8.3 Display-Adaptive Window Management

Previous research has shown that window management on very large displays is a mentally demanding task for the user. Considering that the number of application windows increases on large displays [112], Bi and Balakrishnan [32] observed that users spend significantly more time on window management on large displays than on conventional monitors. In particular, users spend much more time moving and resizing windows. On conventional monitors, minimizing and maximizing windows is a more frequent activity.

Researchers have investigated how users organize their windows on large displays. Findings from these observations indicate why users invest this increased amount of window management on large displays. For instance, Grudin [91] showed that users rarely span application windows across multi-monitor bezels. Instead, the "first" (usually larger) monitor mostly contains windows for the primary task and the second monitor contains secondary resources and communication channels. Although this aspect has never been investigated, we assume that users will also avoid window placement across physical corners on irregular projected displays.

Hutchings and Stasko [113] observed that users are "carefully coordinating" their windows on large displays to keep a small portion of occluded windows visible for direct access. This window arrangement alleviates the need for explicit window switching. However, manually arranging the windows is rather time-consuming and keeping important elements visible for easy window identification requires frequent re-adjustment as other windows are moved.

Bi and Balakrishnan [32] found that users employ a focus and context separation when provided with a large, seamless display without any physical separation. In the peripheral context region, windows provide awareness of particular background information. *Snipped windows* [114] address this issue by showing only window regions relevant for the primary task in context windows.

Current window managers exploit little knowledge of the environment to support the users in manual window management. One of the most sophisticated operations with respect to display adaptivity is the *maximize* operation: With the click of a single button embedded into the window's title bar, the window is automatically resized to the extent of the output device the window is currently residing on. However, the maximize operation is

rarely used when working with very large displays [32, 113]. In addition, it is not suitable for irregular projected displays, where the display boundaries and physical irregularities hardly correspond to the rectangular extents of the output devices (Figure 8.11).

Figure 8.11: Conceptual sketch of a maximized window on an irregular, tiled display: (a) without display-adaptivity, the window is maximized to the extent of one output device while the boundaries are cropped at the irregular display outline; (b) display-adaptive maximize optimizes the window size to the irregular outlines and avoids spanning the window across physical discontinuities. Due to the irregular display outline, oddly shaped display regions remain uncovered by the maximized windows.

Display-adaptive window management is based on the concept of importance-driven compositing (Section 7.4) and automatically created display importance maps (Section 7.3.1) to support users in managing windows on large, irregular displays. Display-adaptive window management supports the user in the following activities:

Display-geometry snapping: as the user moves a window, it automatically snaps within usable display regions, to avoid cropping of window content at display boundaries or placement of windows across physical discontinuities.

Semi-automatic window coordination supports the user in maintaining a "carefully coordinated" window layout [113]: as the user moves a window, overlap with other windows' important content is avoided. If important elements are occluded, either the dragged window or the occluded windows are displaced to keep important window content visible.

Display-adaptive maximize: the user can quickly resize a window, so it covers the maximum visible and continuous display space (see the conceptual sketch in Figure 8.11(b)).

Desktop widgets: desktop widgets, like application launchers, clocks, or notifiers, are automatically moved towards otherwise unused display regions to waste as little space as possible.

These features will be discussed in more detail in the subsequent sections.

8.3.1 Display-Geometry Snapping

Snapping windows to output device boundaries or adjacent windows while moving is already a quite common window management feature. When moving a window, window and output device boundaries act as *sticky edges*, perfectly aligning the window along the edge if it was released within a certain distance threshold. However, current window managers only manage rectangular screen elements. When working with irregular displays, display boundaries and edges are rarely strictly rectangular. In contrast, the display outline is rather polygonal, possibly concave, and physical edges are only approximately aligned along the screen's horizontal or vertical. As an example, consider the display importance map in Figure 7.4.

Display-geometry snapping is activated when the user releases a dragged window. The window layout routine of importance-driven compositing finds the optimal window location with respect to the display importance map. The constraint on maximum window displacement from its original location determines the distance threshold used for snapping. Within this maximum displacement value, the window is re-positioned to be entirely contained within a planar display region and displayable pixels, respectively, if possible (Figure 8.12(a)). As the drag window is automatically re-positioned, the cursor position is also adjusted, so it remains at the same location relative to the drag window.

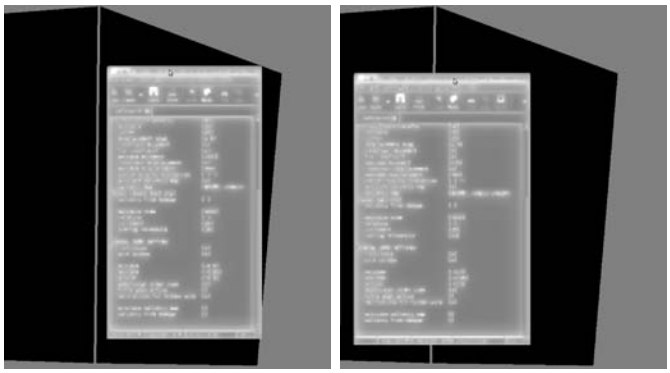


Figure 8.12: Screenshots of the resulting desktop importance map when moving a snap window to a small, irregularly shaped display region (display importance map of Figure 7.4): (a) as long as the window fits within the region, it snaps along the irregular display boundaries, (b) if the window becomes too large, the window is placed along the physical edge, so only little important information is located directly on top of the edge.

The user can choose whether window snapping is applied to the window as a whole (or "black box") or whether the visual content should be considered when snapping a window. In the first case, the information overlap is minimized according to Equation 7.1, so the window is placed to entirely fit within a suitable display region, if possible. In the latter case, information overlap is evaluated using Equation 7.2. The algorithm aims to avoid placement of important window elements along physical discontinuities, even if the

window cannot be repositioned within a suitable region as a whole (Figure 8.12(b)).

User-defined information in the display importance map can furthermore influence the resulting layout. For instance, users could define desktop regions where window placement should be avoided to keep frequently used desktop icons visible [113].

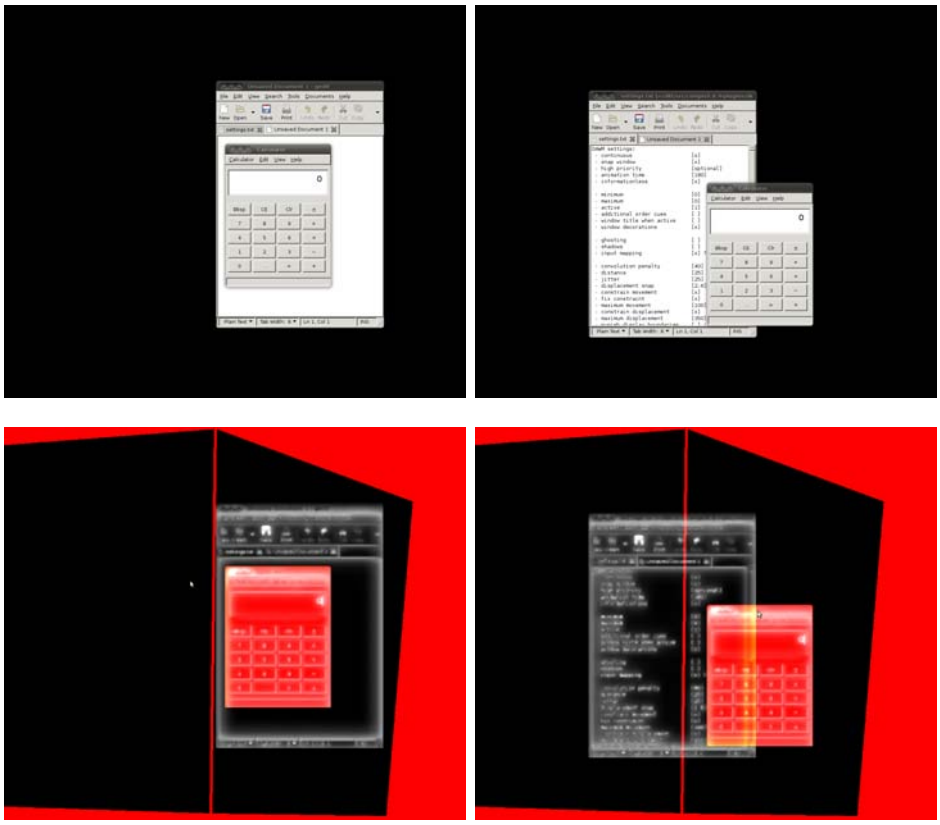
8.3.2 Semi-Automatic Window Coordination

Semi-automatic window coordination should support users in maintaining a "carefully coordinated" window layout [113]. Conceptually, it works similar to *overlap-avoiding dragging* [27]: As the user releases a dragged window, either the underlying windows are re-positioned or the position of the dragged window is adjusted to avoid information overlap. In contrast to Bell and Feiner's overlap avoidance [27], the window layout routine of importance-driven compositing does not treat windows as *full-space rectangles* (or black boxes) but rather considers the desktop importance map as rich information map. Thus, windows will only be re-positioned, if crucial information is about to be covered (Figure 8.13). As a result, important elements of underlying windows remain visible, so the user can easily identify the window and bring it to the front by simply clicking within its boundaries. In contrast to the previously proposed window uncovering technique (Section 8.2), the applied window layout persists and no see-through compositing is applied.

Figure 8.13: Semi-automatic window coordination: (left) the user drags a window towards the left, which causes an overlap with another window. The window layout persists as long as the amount of covered information is low. (right) If more information is occluded, the obscured window is re-positioned to leave the most important window content uncovered. (Window trails were added for illustration purposes.)

We distinguish two operation modes of semi-automatic window coordination: dragged windows can either have high or low priority. In the high-priority mode, the dragged window is added as first window to the desktop importance map and is thus only subject to display-geometry snapping, as described in the previous section. All other windows are subsequently visited and slightly adjust their position, if the released drag window, or any other higher prioritized window, covers important content, as illustrated in Figure 8.14. The penalty for information overlap (ω_i in Equation 7.3) is higher for the drag window than for the other windows, so display-geometry snapping of the dragged window is more aggressive than overlap avoidance of lower level windows.

In the low-priority mode, the dragged window is treated as last window in the window



(a) (b)

Figure 8.14: Moving a high-priority drag window to a small, irregularly shaped display region (display importance map of Figure 7.4): (a) as the underlying window does not have any important content at the drop region, it is not necessary to re-arrange it, (b) if the underlying window shows salient content, it is re-positioned to keep important areas visible. Note how the physical edge is aligned along an area not containing any text content. The top row shows the screen shots, while the bottom row shows the corresponding desktop importance map with the red channel encoding the saliency used for calculating the information overlap for window placement. The green and blue channels encode information for blending and input assignment for see-through compositing, and are not used for display-adaptive window management.

layout routine. Thus, the window layout of the other windows is stable, while the location of the drag window may be adjusted when important content in other windows is about to be covered.

In both operation modes, the top-level drag window is treated as black box, as no see-through compositing is used to reveal occluded content. In the high-priority mode, this implies that lower level windows are re-positioned whenever important content is covered – irrespective whether the window region of the drag window is highly salient or not. In the low-priority mode, the drag window is re-positioned whenever any part of it covers important content in lower level windows.

Treating the drag window with different priorities modifies the rendering order of windows. However, the compositing step of importance-driven compositing takes care that the drag window is always rendered on top of the window stack (Section 7.4.3). To keep the window layout as persistent as possible, penalties for window displacement and jitter (ω_d and ω_j of Equation 7.3) are high.

8.3.3 Display-Adaptive Maximize

So far, importance-driven compositing has only been used to optimize the window locations. However, resizing windows is similarly important on large displays, where windows are rarely maximized [32, 113]. Finding the optimal trade-off between the amount of cropped information at the window's boundary and the amount of occluded information in other windows can be a tedious task. In particular on irregular displays, maximize is no longer feasible, as current window managers are neither aware of irregular display outlines, nor of physical discontinuities.

We therefore created a first prototype of display-adaptive maximize, which takes the physical display properties, but also other windows on the display, into account when finding the optimal window size. As the user issues the command to initiate display-adaptive maximize, the window size is incrementally enlarged in all four directions. For each window size increment step, we retrieve the amount of covered information according to Equation 7.1. This information overlap value is then normalized according to the current window size. If the increase in information overlap, caused by the window size increase, exceeds a given threshold, the window is no longer extended in the given direction. The procedure is repeated until the window cannot be extended in any of the four directions any more, or if all screen boundaries (*i.e.*, the edges of the desktop importance map) have been reached.

Again, we discriminate between high and low prioritized windows by adjusting their rank in the window layout routine. Windows with high priority are only evaluated against the display importance map and will therefore only be limited by physical display boundaries. Lower level windows are either covered or, if their penalty on information overlap is reasonably high, slightly re-positioned to reveal important content. Windows to be maximized with low priority are evaluated against the entire desktop content and thus avoid overlap of other windows, as illustrated in Figure 8.15.

The incremental resizing process is currently implemented as CPU prototype. Each resizing step is performed as Xlib function call. After each resizing operation, a re-paint is triggered to retrieve the current information overlap value by the window layout routine (Equation 7.1), implemented in OpenCL. Thus, the maximize process can take up to several seconds, depending on the display size and chosen step size. The current display-adaptive maximize functionality, as described above, could be entirely ported to OpenCL, so no delay would be noticeable for the user.

The advantage of the CPU approach facilitating Xlib resizing is that we can assess

Figure 8.15: Screenshots of the resulting desktop importance map when maximizing a low-prioirty window in an irregularly shaped display region (display importance map of Figure 7.4): (a) the window layout before the maximize process, (b) and the resulting window layout, where the window to be maximized is squeezed between the calculator and the physical room corner.

the amount of revealed information when the window is actually enlarged (as opposed to simple texture-based scaling on the GPU). Conceptually, this information could be used to assess the amount of information revealed in the previous resizing step. In this way, display-adaptive maximize could be extended to *content-adaptive maximize*, which additionally evaluates whether it is actually useful to further enlarge the window, even though there is still enough display space available. Figure 8.16 illustrates this concept.

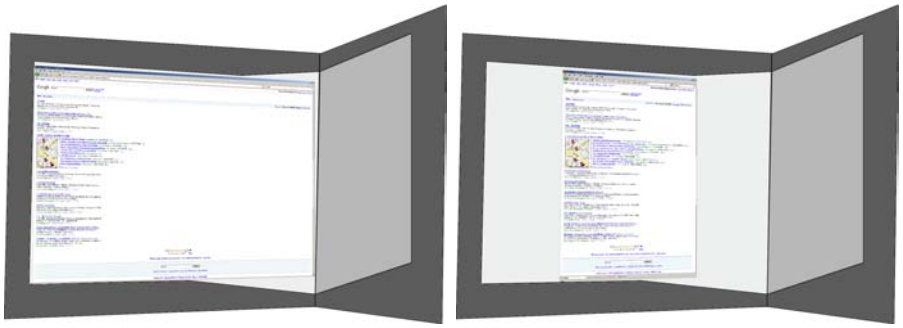


Figure 8.16: Conceptual sketch of content-adaptive maximize: (a) display-adaptive maximize finds the maximum window size with respect to the display outline; (b) adding content-adaptivity would furthermore restrict the window size if no more important information can be revealed when the window is enlarged.

However, simply determining the visual saliency in the window to be maximized by $\sum_{(x,y)\in\Omega_w} I_w(x,y)$ was found to be a poor measure to assess the overall amount of important information. We observed that GUI elements, such as menu bars, contribute a high saliency, while adding simple content, like text, is hardly measurable. In the future, alternatives to assess the overall window importance are therefore necessary. It may be sufficient to automatically determine a window's content pane and to apply the window

importance evaluation solely on this window region. It has to be noted, though, that automatically determining the content pane is only possible based on an image-based segmentation of the window texture on window manager level. In the X Window System, only a few windows consist of a hierarchy of X windows that can be accessed by the window manager – depending on the employed GUI toolkit. A truly reliable measure is probably not possible by image-based analysis of window textures and requires additional information provided by the applications themselves – like the *zoom* function of OS X, which toggles between a user-defined window size and a *standard state*, defined as the optimal window size by the applications.

8.3.4 Desktop Widget Layout

When using irregular displays, the display outline is no longer rectangular. In contrast, a complex polygonal outline with strong concavities may arise. These non-rectangular outlines make it difficult to place common desktop elements, like window lists, application launchers, or notifiers. In current windowing systems, these elements are usually arranged in menus along the display boundaries and therefore will be partially lost when cropping the screen along its irregular outline (Figure 8.17). In contrast, the concave regions at a display's boundary remain mainly unused by conventional desktop elements, like windows or menus, which are simply too large to be displayed within such as small area (*cf.*, conceptual sketch of Figure 8.11(b)).

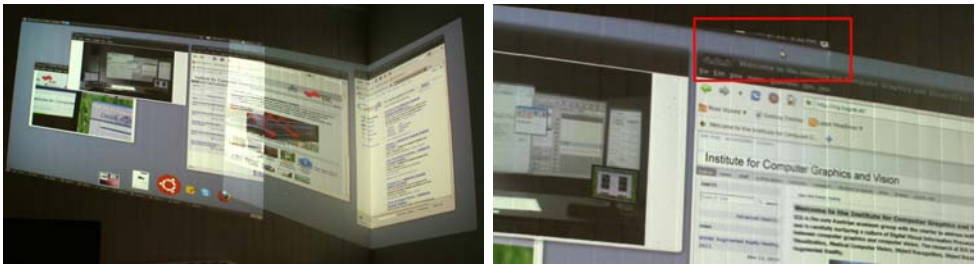


Figure 8.17: Conventional desktop widgets (menus, icons) on an irregular display: (a) without geometric compensation, all icons and menus are visible but (b) our geometric compensation approach crops peripheral display areas, so these desktop elements are almost entirely lost.

As an alternative, we are using free-floating desktop widgets, which are increasingly popular in conventional windowing systems, to substitute the desktop menus along the screen boundaries. Prominent examples for desktop widgets are analogue watches or small calendars. On a conventional display, maximized windows typically cover such desktop widgets. As an alternative, the user may keep them on top of application windows, thereby occluding potentially important window content. With display-adaptive window management on irregular displays, desktop widgets are subject to semi-automatic window coordination. Thus, the small low-priority widgets are gradually moved towards very small display areas where no other desktop content is likely to be positioned (*cf.*, Figure 8.18).

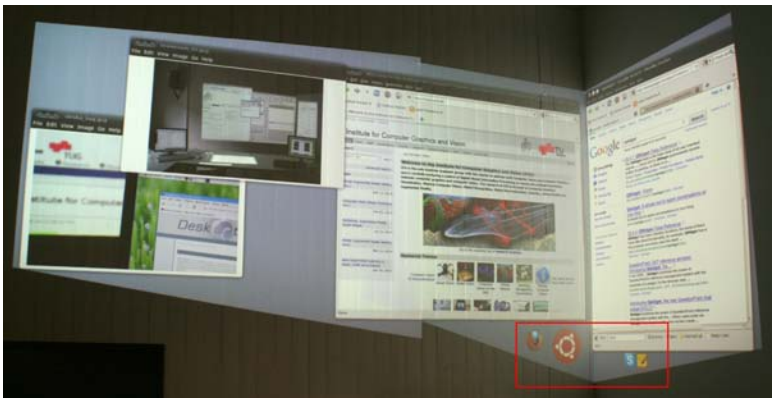


Figure 8.18: Desktop widgets on an irregular display are subject to semi-automatic window coordination and will therefore be moved to peripheral areas where no larger items can be reasonably placed.

8.4 Polarization-Based Interfaces

Large displays afford the visualization of a vast amount of data and multiple interacting users, as there is usually enough display space and physical space in front of the display to coordinate multi-user activities. Visualization of complex datasets is often addressed with multiple coordinated views on such a large display space, so multiple users can view the data in their preferred visual encoding while maintaining awareness of the other user's visual representation. A large display offers ample space for arranging multiple views side-by-side. However, such a spatial separation of visualization views requires the user to mentally merge corresponding elements from multiple views, or requires dedicated visual linking techniques (see, for instance, Section 8.1).

An alternative to cue-based linking techniques to visually highlight related elements across multiple instances is to use magic lenses [38] as an in-place display for information filtering. A magic lens is a user-controlled widget which provides a filtered view of the content underneath. Of particular interest for large displays and multi-user operation are tangible magic lenses which allow the user to change the magic lens region on the underlying data by physical movement of the lens device. Mind that this approach is the only concept presented in this thesis deviating from a strict WIMP approach, as interaction does not rely on pointing devices but uses the rich capabilities of a tangible user interface instead. We will show (conceptually) that this interaction technique can be employed in combination with conventional WIMP interfaces.

However, there are several limitations that make the employment of tangible magic lenses unfeasible. First, both passive and active magic lens devices suffer from disadvantages in operation (*cf.*, Section 2.1.1): Passive magic lenses alter the underlying imagery and thereby potentially interfere with collaborators' activities. Ideally, each user would be equipped with an active magic lens (*e.g.*, in form of a tablet PC) to unobtrusively filter the displayed visualization with personalized information. However, such a kind of active

magic lens is quite cost-intensive and also requires a sophisticated distributed software infrastructure.

Second, both categories of tangible magic lenses require accurate 6-DOF tracking to reliably determine the filtered viewpoint on the shared dataset. Accurate 6-DOF tracking is again very cost-intensive.

Finally, conceptually, magic lenses alter a shared view on a display and provide personalized in-place filtering of this view. In some situation, the opposite approach may be desirable: The user sees a personalized view and can use the magic lens to locally display a shared view.

Polarization-based interfaces are a low-cost alternative to active magic lenses for collaborative information management. By exploiting properties of polarized light, users can choose between two personalized views of the environment. A purely optical and tracking-less magic lens can be used to locally filter the personalized view. The optical magic lens approach thereby eliminates common problems of passive or active tangible magic lenses.

8.4.1 System Description

The hardware setup consists of two commodity projectors as used for a conventional passive stereo projection system. In such a system, two projectors create superimposed images on a polarization-preserving screen. The projectors receive two different output images (the two stereo views) and are equipped with a horizontal or a vertical polarization filter (HP and VP), respectively. The viewer wears inexpensive glasses fitted with an HP filter for the left eye and a VP filter for the right eye, to obtain proper channel separation for the two eyes.

In contrast to such a standard passive stereo setup, viewers of our system either wear HP or VP filter glasses, so under normal viewing conditions, only one of the two projector images is visible. In this way, we can separate two content channels: the primary view, which the user sees through her glasses, and the secondary view, which is blocked.

This setup results in a *single display privacyware* [219], where private information can be visualized in the dedicated channels only, while shared information is displayed in both channels. In its original implementation, single display privacyware [219] relied on an active stereo approach to separate the channels: It used active shutter glasses and frame-interleaved display of the two privacy channels. By opening both glasses with the synchronized display of one channel and closing the glasses with the subsequent one, the second channels is effectively blocked for the user. Theoretically, more than two channels can be separated with this approach, but with an increasing number of users flickering will be perceivable. Our use of a passive stereo system strictly limits our system to two channels only. In addition, users have to look relatively straight towards the display surface and may not tilt their head much in order to preserve the linear polarization.

The main reason we employ a passive stereo setup for channel separation is the possibility to employ optical magic lenses for inexpensive tangible filtering. Optical magic

lenses exploit the unique property of liquid crystal display (LCD) panels: Without power applied, they are designed to retard linearly polarized light by 90°. In conventional LCD monitors, this property is used in combination with a polarization filter to control light flow through the panel by applying power selectively to portions of the LCD panel to make selective pixels either transparent or opaque.

By using passive unpowered LCD panels as 90° retarders, the channel separation is effectively inverted through the panel in a polarization-based projection setup. Polarized light from a projector passing through a retarder changes from HP to VP, or vice versa. Figure 8.19 illustrates this approach.

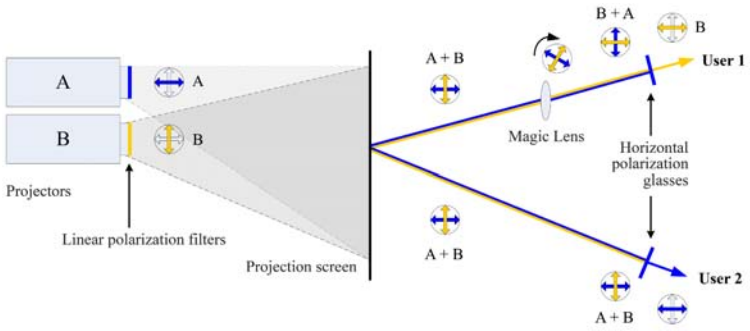


Figure 8.19: System sketch of polarized interfaces and optical magic lenses: Linear polarization filters (HP for projector A and VP for projector B) create polarized super-imposed images on a polarization-preserving projection screen. Both users are wearing HP filter glasses and therefore see projector A's image as primary view. User 1 uses an optical magic lens to filter the image. Thus, projector B's image changes to HP within the lens and therefore passes the HP glasses instead.

A similar effect can be obtained with cellophane [116], but our experiences have shown that retardation is typically below 90°, leading to ghosting effects. The degree of rotation of polarized light depends on the wavelength. Thus, polarization rotation varies with different colors and results in a color shift, depending on the used retarder material. When using the LCD lens, this effect is hardly perceivable. Figure 8.20 shows a cellophane magic lens filtering a map display. Mind that the lens is not tracked and that the shared view is not altered by this operation.

For properly aligning the two superimposed projection images, we applied a simple manual calibration routine in our prototype implementation. However, correct superimposition can be easily created automatically by the display environment, using a similar calibration approach as presented in Section 4.1.

8.4.2 Application Areas

Our prototype software implementation relied on a simple image viewer, showing two different images on the respective output devices. Therefore, the subsequent discussion has to be seen as purely conceptual.

Figure 8.20: A satellite map view is locally filtered for a street map view.

Application areas of polarized interfaces in combination with optical magic lenses are manifold. As a prominent example, all spatially aligned dual-view coordinated visualizations can benefit from this interaction technique to support direct visual comparison. Examples are superimposed map views, as shown in Figure 8.20 or XRay vision, as illustrated in Figure 8.21. In practice, we support the same visualization types as other systems superimposing spatially aligned images for direct comparison, such as *LivOlay* [127].

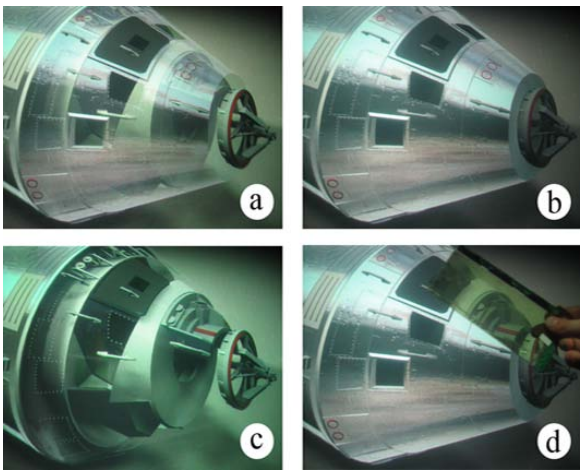


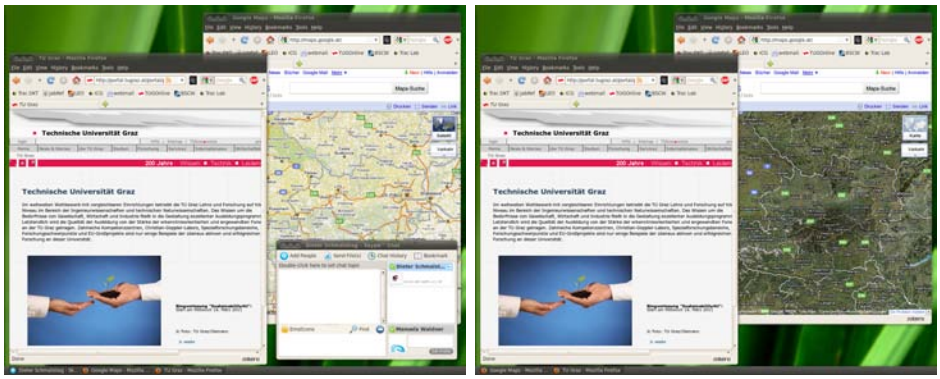
Figure 8.21: XRay vision using the optical magic lens: (a) the superimposed channels without polarization filter, (b) the first and (c) the second user's primary view, and (d) the first user's view filtered using an LCD magic lens. (3D model by 3D studio max)

Our system can also be employed to add details on demand by selectively adding a more detailed secondary view. For instance, the secondary view may show an annotated version of the primary view. The magic lens is then employed to locally add the annotations.

When using our polarization-based interfaces and magic lenses, window management

has to be substantially changed to adapt the content to this situation. Using our system infrastructure, the window manager deals with two equally sized output devices, which correspond to the two super-imposed projectors and output channels, respectively. When used as single-display privacyware, as proposed by Shoemaker and Inkpen [219], the window manager needs to differentiate between shared and protected application windows. Shared windows are duplicated to each output device, while protected windows are displayed on the respective user's channel only. Private screen information, such as the user's cursor or personalized visual links (*cf.*, Section 8.1), is also limited to a single channel. To make private information apparent to the other user, the personalized information can either be shared on software level (*i.e.*, the window manager duplicates the information to the second output device) or the collaborator uses the magic lens to switch to the secondary view.

Applications aware of the privacyware setup can create window duplicates for themselves, containing alternative views of their displayed content. Examples for maps and XRay vision visualizations are given in Figures 8.20 and 8.21. Instead of rendering the window twice, the window manager then manages the secondary window provided by the application as window duplicate, which is spatially arranged according to the primary window. Figure 8.22 shows a mock-up of the window manager concept for single-display privacyware with two output channels.



(a) (b)

Figure 8.22: Screen mock-up of window management for single-display privacyware [219] window management: The window manager controls two output devices, one for each projector (a and b). The browser window is a render duplicate, while the map window is a coordinated window duplicate by the map application itself, so each user can use the preferred visualization type. The chat window in (a) is protected for user 1. The individual channels can be furthermore enhanced with personalized cursors and visual links.

8.5 Discussion

The presented window management techniques are first steps towards supporting the users in information management in emerging display environments. Summarizing, the challenges the user faces in emerging display environments, as compared to conventional desktop computing, are:

- an increased amount of display space and number of pixels to operate,
- an increased amount of visible information, as more application windows can be arranged side-by-side,
- still a considerable amount of occlusion due to an increased number of open application windows,
- the possibility to conduct shared information management due to increased display space and physical space in front of the display, and
- irregular displays containing merely rectangular user interface elements.

With conventional window managers, these properties lead to an increased window management and coordination effort for the user.

The proposed window management techniques are designed to assist users in discovering information on large, cluttered displays (visual links), to discover and access potentially occluded information (uncovered windows), to manage a large amount of windows on a large, irregular display space (display-adaptive window management), and to share information among collaborators with little interference (polarization-based interfaces and collaborative information linking). However, there is a much larger design space to explore in the context of window management. Our techniques relied on additional information rendered on top of application windows, modifications to the spatial window layout, and modifications to the window textures' appearances (*e.g.*, transparencies). Aspects that could be furthermore explored are geometric deformations (*e.g.*, context-sensitive scalings, similar to [9]) and further appearance adjustments, like color or shading effects.

Our presented techniques bridge information gaps between windows by visually connecting related items and soften the rectangular window layout by allowing context-sensitive "holes" in windows. However, we did not completely "break out of the box". In the future, information chunks may be automatically cut out from their application and "window" managers may treat these chunks as more or less independent screen entities, which may or may not be semantically connected to other entities – irrespective of their parent applications. With such an approach, screen real estate can be facilitated much more efficiently, reserving visible space only for items of current importance. However, major changes to current application development and window management approaches are necessary, to allow for a rich central application coordination and sufficiently reliable classification of content importance.

Regarding display form factors, we illustrated how display-adaptive window management could enhance management of conventional, rectangular application windows on tiled and irregular displays. However, display-adaptive window management only covers a small aspect of potentially emerging display form factors. For instance, window management for very small devices or displays with ambiguous orientation (*e.g.*, tabletop displays) are still open issues. Window management for those displays again needs substantial extensions to the underlying windowing system to support basic operation. For instance, the compositing window manager itself can easily apply arbitrary transformations to a window, which is treated as simple textured quad in a 3D scene. Thus, rotating a window on a tabletop display is an easy task from the window manager's point of view (Figure 8.23). However, the underlying windowing system, which is responsible to handle input events, is restricted to upright oriented, rectangular windows. Any deviation of the rendered window from this simple internal model leads to an offset between viewing and interaction space. Thus, we do not only need flexible window managers. For the creation of truly innovative window manager techniques, the underlying windowing system approach needs to be fundamentally changed. Until then, window manager techniques will build upon work-arounds (or one might even call them "hacks"), such as presented in Chapter 7.

Figure 8.23: Rotated windows on a tabletop display: adapting the viewing space in the window manager is easy, but the interaction space of the windowing system still relies on upright oriented windows.

In the context of multi-display environments, an additional aspect is worth investigating: migration of windows between heterogeneous displays. As an example, the user may move a window from her smart phone display to a very large wall display, and subsequently to a tabletop display. How does the visual representation of the window change as it is migrated between the different devices?

Currently, window managers coordinate window content as textures, which are mapped onto 2D quads. In addition, the above mentioned restrictions by the windowing system to upright oriented windows apply. As a result, the only operations a window manager

can reliably support are: moving and resizing, as well as image-based operations on the individual windows or the entire desktop. However, these operations will not be sufficient in such a complex environment. Heterogeneous display form factors require substantially different representation of the content – differing in the (semantic) zoom level of the displayed content, the layout of the window's content, and the amount of content that has to be cropped due to screen size limitations.

With current windowing systems, these operations go far beyond window manager capabilities and instead require adaptivity of the applications themselves. To provide application developers with a rich variety of possibilities and information about the environment, Myers *et al.* [161] requested extended support for ubiquitous computing environments in GUI toolkits already more than ten years ago. GUI toolkits could provide API calls to request information about the environment (*e.g.*, display form factors and user arrangements), multi-user event notification, and provide pre-defined sets of basic UI elements that automatically adapt their appearance as the associated display form factors, the users' viewing arrangements, or the interacting user changes. Several concepts and implementations of such *adaptive user interfaces* have been presented in the past. For instance, *SUPPLE* [86] generates user interfaces by evaluating an optimization problem which satisfies device constraints (*e.g.*, size) and minimizes the estimated user effort. With such a foundation, application developers can easily develop fully environment-adaptive applications, while windowing systems take care about the actual window migration, window layout management, and multi-user input support.

Chapter 9

Evaluations of Window Management Techniques

The window management techniques presented in the previous chapter all incorporate knowledge from the environment layer or the window layer – or both. According to the hypothesis of this thesis, awareness of the physical environment and window content will support users in their information management activities. The purpose of the subsequent experiments is to evaluate the proposed techniques with respect to this hypothesis.

The first experiment (Section 9.1) was an exploratory, observational analysis of a complex information analysis task. Users were provided visual links across applications (Section 8.1) to solve the task. The aim of the experiment was to initially asses whether visual links across applications supported users in information discovery.

The second experiment (Section 9.2) was an exploratory, observational analysis of a complex information analysis task, conducted by a team of two users. Users were free to use personalized visual links across applications and supportive features (*e.g.*, the selection bookmarking tool) to support the analysis task (Section 8.1.2). This experiment was conducted to get some preliminary experiences with visual links for information sharing. We also gathered observational experiences with co-located collaboration on single-display groupware settings.

The third experiment (Section 9.3) compared three window management techniques to discover and access occluded window content. We compared task performance of three interfaces: conventional Alt+tab window switching, *free-space transparency* [123], which applies content-aware transparencies to occluder windows but does not optimize the spatial window layout, and our window uncovering technique (Section 8.2).

Finally, the fourth experiment (Section 9.4) was an exploratory study comparing window management on an irregularly shaped projected display with simple display-adaptivity features to purely "manual" window management, which is unaware of the physical display form factor.

9.1 Exploratory Evaluation of Visual Links across Applications

We conducted an informal user evaluation to assess the subjective user acceptance and the benefit of visual links across applications (Section 8.1) for information discovery with complex information sources.

9.1.1 Participants

Seven participants (aged 25 to 39, 2 female) participated in the study. They were recruited from a local university and a software development company. Five users are working with a single-monitor setup, one uses a dual-monitor setting, one user is working with three monitors on a regular basis. Except for one user, all participants use monitors with 1680x1050 pixels or more on a regular basis.

9.1.2 Setup, Task, and Procedure

Users had to conduct a complex information analysis task on a 26" monitor with 1920x1200 pixels. Information was spread to four different application windows, as shown in Figure 9.1. The map application on the left was showing a static view of the African continent. On the right, we placed a *Caleydo* [142] window, showing demographic and economic statistics of 190 countries in a parallel coordinates view. In between, an HTML-page in a web browser listed all African countries exporting oil, while a second web browser window below listed all African countries. Prior to the actual task, users were trained on parallel coordinates and how to invoke visual links from the respective applications.

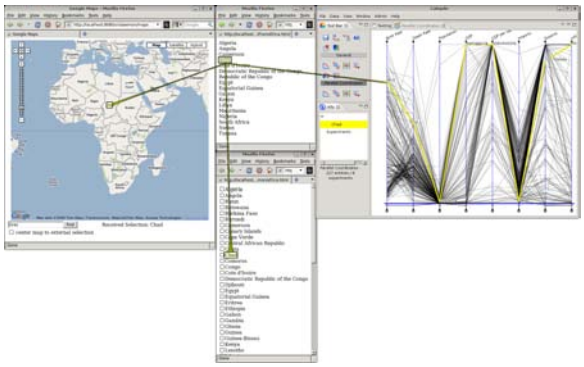


Figure 9.1: Window arrangement for the user evaluation of visual links across applications: a map application fixed to Africa, two websites with a list of countries, and a parallel coordinates view of demographic and economic statistics of 190 countries. Visual links visualize correspondences between the individual windows.

The task was to find all African countries north of the equator, which export oil and have a birth rate (*i.e.*, number of childbirths per 1000 people per year) higher than 30. The birth rate data was plotted on the first axis in the parallel coordinates view. Countries fulfilling these requirements had to be marked on the HTML-page containing

the list of all African countries. We chose the example of Africa, as demographic and economic statistics were strongly diverse across the countries and we expected our users to be relatively unfamiliar with the exact geographic locations of African countries. We measured the task completion time, recorded observations, and collected user comments in an interview.

9.1.3 Results

Except for the task completion time, we did not gather any quantitative measurements. Obtaining quantitative results with such a complex task is quite difficult, as performance measurements greatly differ among participants. We will therefore report on observational evidence and user feedback.

Observations

All but two users could successfully complete the task with no errors. They required 1:40 to 4:30 minutes to complete the task.

One user did not succeed as she missed the small country Equatorial Guinea, which is actually located north of the equator but hard to identify on the map with the applied zoom level. Two other users had to zoom the map to determine the exact location of the country. One user mentioned that selection regions are not optimally placed in the map application. As the Google Maps API only delivers the geographical location of the center of the country and corresponding screen coordinates, we do not have control over label size and do not have the possibilities to highlight the actual country outline. Instead, we use a small bounding rectangle of fixed size to visualize the geographic location. As a consequence, the size of the region does not indicate the size of the country. Additionally, connection lines often cover the label of the country, which is – despite transparencies – hard to read.

One user quit the task after four minutes without finding any country fulfilling the requirements. Contrary to all other users, she mainly employed the parallel coordinates view to invoke visual links. However, as the parallel coordinates contained data from 190 countries, it was tedious to find the polylines for the required African countries. All other participants used the browser window with the list of oil-exporting African countries as source window for visual links, as the information was already strongly filtered with only 17 potential countries. From this window, they simply had to follow the link to the parallel coordinates and check the birth rate and the link to the map to see the geographical location. Finally, they followed the link to the browser window containing the list of all African countries to mark the result. The fastest participant could correctly solve the task in one minute and 40 seconds using this strategy.

We generally observed difficulties when users employed the parallel coordinates view as source window. Apart from the one user not finishing the task, we noticed four other users moving the mouse pointer to the parallel coordinates after selecting the country in

the browser window. As selection was triggered by mouse-over in the parallel coordinates, their previous selection was lost when the mouse pointer was accidentally moved across a different polyline and they had difficulties finding the polyline for the previously selected country again.

Subjective Feedback

User feedback was consistently positive, and was also helpful in terms of identifying user interface problems. Users described working with visual links as *"It just works and it is efficient"* and *"It is fun"*. All users agreed that having coordinated windows with visual links is helpful for such a task. They stated that it was clear what visual links visualize and how items were related. Distraction caused by the large amount of visible information was reduced as *"it shows only the most essential information"*. One user noted that *"otherwise [without visual links], the task would take ten times longer"*. Except for the one user not being able to find any countries, all users agreed that task completion was very efficient using visual links. One user explained he was fast as he would simply trust the visual links and would not check whether the correct items were connected. Another user said he always double-checked the connections. Surprisingly, this participant completed the task fastest. However, some users doubted whether visual links are useful for every-day work (for software engineers).

Several users commented on the interaction techniques provided by the respective applications. As we could also observe during the task, accurate selection in the parallel coordinates view was tedious – in particular with this large amount of information available. Invoking visual links by moving the mouse pointer across a densely populated parallel coordinates view helps to get a quick impression of the data but is not suitable when accurate selections are required. Two users also noted that selecting text in the browser window is complicated, especially if multiple words need to be selected. One user suggested employing an "auto-invocation", which automatically triggers visual links shortly after a word has been selected.

Another user demanded multi-selections, so he could select multiple countries in the browser and see the respective connections to the visualization software and the map application. With multi-selection, users could have selected all oil-exporting African countries in the first browser window, where each of these selections would have received their own visual links. This feature would reduce the required user intervention, as invoking visual links is required only once. However, it is subject to further research whether the additional line connections introduce so much visual clutter that extracting related information becomes more tedious.

9.1.4 Discussion

This initial and informal user observation and feedback encourages our approach of using visual links across applications to support users in information discovery. Users reported

that they were subjectively fast and could more easily distinguish important from unimportant information. Swaminathan and Satoh [234] reported that distinguishing important from unimportant information is one of the greatest challenges in user interaction on very large displays, where a lot of information is visible simultaneously.

In the future, it will be important to evaluate the benefit of the visual links' form factor for information discovery. For that purpose, a formal user experience comparing simple frame-based highlighting with visual links and an advanced form of visual linking* for a visual search task on complex visualizations and images will be conducted.

As also noted by our users, visual links are not supposed to be a permanent feature of the window manager. Instead, we envision visual links to be an on-demand feature that can easily be switched on for information analysis and comparison tasks.

Users also raised some ideas for improvements and commented on several deficiencies of the interaction with visual links. Indeed, visual links provide a consistent visualization across multiple applications, but as the user interaction technique to trigger a selection is provided by the applications themselves, there is no consistent way to create links. According to the user feedback, it is important to provide an explicit technique to trigger links (as opposed to the parallel coordinate view, where links were created on mouse-over), which is consistent across all applications, such as a keyboard shortcut. Technically, this could be easily solved on window manager level: as the user selects an item, she presses the keyboard shortcut, which is captured by the window manager. The window manager determines the active window and the management routine contacts the associated application to trigger a user selection. If no application is associated with the active window, the selection is treated as one-shot selection (see Section 7.2.1 for more implementation details).

9.2 Exploratory Evaluation of Collaborative Information Linking

We conducted an exploratory evaluation of co-located collaborative information analysis on a single-display groupware [226] setup. Users were cooperating on a large projected wall display, where each user was equipped with a separate mouse and keyboard pair. To solve a complex analysis task, users could use personalized visual links across applications and collaborative storage facilities utilizing the central application coordination routine (Section 8.1.2).

We decided to gather observational evidence as an initial evaluation step, as there is no clear baseline condition for a formal, comparative experiment. We believe evaluating collaborative information linking against a system without personalized visual coordination would not have been a fair comparison. In addition, obtaining quantitative results

*This work on *Context-Preserving Visual Links* was conducted by Markus Steinberger, Manuela Waldner, Marc Streit, Alexander Lex, and Dieter Schmalstieg and has not been published yet. The advanced form of visual linking aims to preserve visual context by avoiding occlusion of perceptually important image content using a similar principle as importance-driven compositing presented in Section 7.4.

for a complex analysis task is difficult, as task completion times and correctness of the results can be influenced by a lot of different factors.

9.2.1 Research Questions

The aim of the experiment was to initially assess the following research questions:

Q1 *How does a pair of users coordinate information access and manipulation on a shared wall display using multi-input support and conventional single-user applications?*
There is a large body of related work assessing co-located collaborative interaction on single-display groupware. An overview can be found in Section 2.2.1. Generally, interaction on a shared vertical display may result in asymmetric interaction [74, 152, 204], unless each user is equipped with a personal input device [41, 245, 246]. The large available screen space is usually spatially separated to create personal *territories* [215, 240, 245], where access restrictions are socially coordinated. This social coordination can be supported by system-imposed access restrictions [159, 178]. Our system did not impose any access restrictions on input devices, but application windows could be locked from other users' links. Normally, these observations are conducted using specialized groupware applications or pen-and-paper prototypes. The core emphasis of this research questions is to assess the nature of multi-user coordination when working with conventional single-user applications and normal WIMP interfaces.

Q2 *How does collaborative information linking support (or impede) users in information discovery and information sharing?*
Our previous experiment indicates that visual links support single users in information discovery. However, it is unclear whether information discovery is still efficient when multiple sets of links are rendered on the screen and how much interference individual information discovery supported by visual links causes for the collaborators. Related research reported that users were even distracted by other users' cursors [258], so a certain amount of distraction can be expected. On the other hand, we assume that visual links support users in resolving deictic references (and thereby in sharing their information), which has been reported as problematic on shared wall displays [119, 204].

Q3 *How does the central application coordination approach and selection storage support users in information sharing?*
Previous research suggests that synthesis of individual results from different collaborators into a compact format is an important aspect of collaborative sensemaking [102]. It has been observed that groups with parallel input support generated much more structured reports than those having to take turns [185]. Prante *et al.* [185] speculate that these detailed reports are necessary to synchronize activities when a lot of parallel work is conducted. Similarly, Mark and Kobsa [153] discovered that users "retraced" their analysis steps when validating their answers collaboratively. Ellis *et al.* [72] also observed that users produced outlines summarizing the individual contributions at the end of a collaborative session. In some cases, users prefer pen and paper or a simple text editor to sophisticated

annotations tools built into a groupware application [3]. In our experiment, users were free how to share and manage their findings. They were provided with a simple text editor, our selection bookmarking tool (see Section 8.1.2), as well as pens and paper. The most interesting aspect in contrast to previous work was to investigate how central application coordination helped to generate reports or outlines, and how collaborative information linking supported users in presenting these reports to their collaborator.

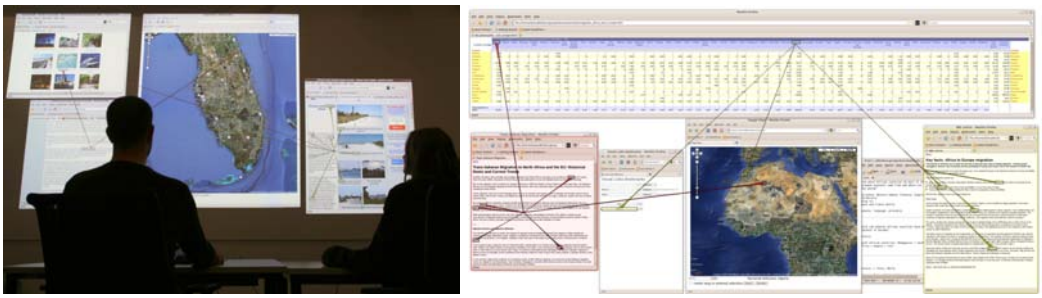
9.2.2 Participants

We recruited eighteen participants (16 males, 2 females, aged 23 to 38) from a local university institute. Only four participants stated to conduct collaborative information analysis on a regular basis, such as investigations on a particular topic, literature research, or research in general, as well as teaching. Half of the users are working with a dual-monitor setup regularly, three with three monitors, and four with a single monitor. All participants indicated to use monitors with 1600x1200 pixels or more (as their primary monitor).

Users had to work in pairs on the information analysis task. As participants were recruited from a single institute, most of them knew each other or were even regularly working together.

9.2.3 Setup

The experiment was conducted on a front-projected display driven by six XGA projectors, resulting in a resolution of 3072x1536 pixels. Projectors were operated by a commodity PC running Linux Ubuntu 9.10 with a dual-head graphics card and two graphics splitters[†]. Each participant had a mouse and keyboard pair to operate the display. Figure 9.2(a) shows the setup.



(a) (b)

Figure 9.2: User study setup for collaborative information linking: (a) the shared projected display with two separate mouse and keyboard pairs and (b) screenshot of the shared display with the prepared application windows and user-specific visual links.

[†]`http://www.matrox.com/graphics/en/products/gxm/th2go/`

We used multi-pointer X [115] in combination with our multi-pointer window manager extensions (Section 7.1) to support concurrent multi-user interaction. Mind, however, that concurrent input on a single application window is possible, but not coordinated, using this approach.

9.2.4 Task

The task was to analyze migration from Africa to Europe by evaluating different sources of information. As information resources, we used content freely available on the internet, which was only slightly adapted to reduce the task complexity. Participants were presented a table showing numbers of migrants from African countries to European OECD countries in 2006[‡], a map, and several articles on the topic[§]. They were asked to answer three questions on African migration to Europe, where each of the questions were divided into focused and open-ended sub-questions. A few questions could be answered by evaluating solely the table (for instance: *"From which African countries do most of the European migrants come from and where are they going?"*), while others required support of the map and background information from the articles (for instance: *"Which African countries are used as hub for migration to Europe? Where are the typical routes?"*).

As application resources, we provided two modified applications, as described in Section 7.2.1: a Firefox web browser and a Google maps mash-up. In addition, a bookmark window and a simple, unmodified text editor (gedit) were available to organize findings. The table and the map were horizontally centered on the shared display. The articles were presented in two browser windows – one for each participant – and located on the respective closest display locations to the users. The two browser windows were protected from the other user's links. The bookmark window was placed on the left, and the text editor on the right display half. Figure 9.2(b) shows the window arrangement.

9.2.5 Procedure

Prior to the task, participants had an approximately 10 minutes warm-up period where they could get familiar with the system and the collaborative information linking technique. They were instructed how to make selections (from the specific applications and one-shot selections), switch off their links, and how to protect and release windows. Task time was limited to 30 minutes.

Groups could decide for themselves how to organize their findings and were free whether to use visual links. We employed system logging, video-taped the session, handed out questionnaires, and conducted a semi-structured interview at the end of the experiment.

[‡]simplified from `http://stats.oecd.org`

[§]shortened from `http://news.bbc.co.uk/`, `http://www.migrationinformation.org/`, and `http://www.wikipedia.org/`

9.2.6 Results

We will report on observations during the experiment and video analysis of the sessions, as well as selected questionnaire results (7-point Likert scale), interview feedback, and logging numbers. For each pair, we obtained about thirty minutes of video material and two pages of field notes taken during the experiment. Based on the field notes and video material, we established coding categories for information linking, such as whether participants were linking for individual information retrieval or whether bookmarks were selected for individual information query or for collaborative discussion.

Collaborative Information Linking

Information linking was used 9-84 times (for different selections) per group. The median number of selections was 32 per group. Information linking usage varied with the group's configuration and work style (*cf.*, Figure 9.3): We observed that users working in a mixed-focus collaboration style [94] (*i.e.*, often switching between individual and tightly coupled work) used collaborative information linking most frequently. Groups working either tightly together all the time or where a single user took control while the other one was merely passive, used information linking least.

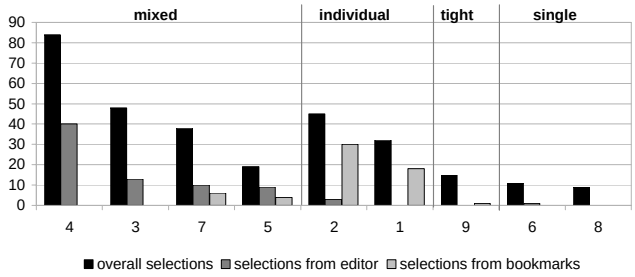


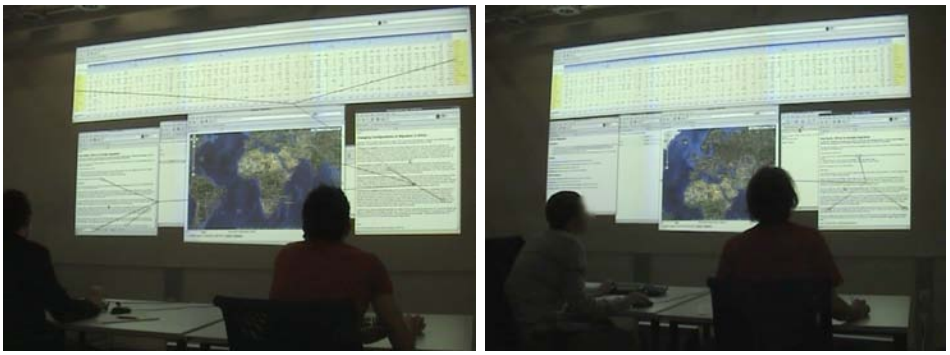
Figure 9.3: Recorded activities for each group sorted by work styles and frequency of information linking: mixed-focus collaboration, individual task-solving, tight collaboration, and a single interacting person (view coupling [240]) were observed work styles. The overall number of selections for information linking, and selections from text editor and bookmark list, respectively, are listed.

When working individually, information linking was mainly employed to locate occurrences of a certain country and to quickly look up its geographic location and associated migration numbers in the table (*cf.*, Figure 9.4(a)). In the post-test questionnaire, the median rating for having visual links *across* application windows on a 7-point Likert scale was 6. Having links *within* application windows was rated 5, as well as having arrows to indicate items outside the visible window boundary. Linking was not actively used when reading articles or scanning the table individually. Links often remained faded during these activities, but were rarely completely disabled. Visual clutter introduced by linking and by the amount of visible application windows was both rated 3.

In seven groups, we could observe situations when users silently watched their partner's

activities while linking. Being able to see links by the partner had a median rating of 6. Users largely agreed that individual links were easy to identify (median score 5). The main distinguishing aspect was the unique color-coding (6), while the direction of the links played a minor role (4). Two groups mentioned that links were hard to distinguish when faded. One user explained he did not actively distinguish his links from his partner's but rather followed his own links from the current window to the relevant target windows – color did not matter for him.

The median score for *I was often distracted by my partner's actions* was 2.5. In the interview, users mentioned that interference and distraction was mainly caused by changes to the window layout caused by the other person, as well as concurrent interaction on shared application windows. Only one user mentioned that he was *"a bit annoyed"* by his partner's links. The ability to protect windows from other users' links was rated fairly low (median score of 4.5). It has to be noted that the browser windows containing the articles were protected when the experiment started. Only two users switched window protection off during the task, but no user locked one of the initially unprotected windows.



(a) (b)

Figure 9.4: Video stills from the experiment: (a) individual information retrieval supported by personalized links and (b) tightly coupled collaboration using a single set of links.

When working in a tightly coupled fashion, one participant usually contributed linking while the other set of links was either faded or switched off (as shown in Figure 9.4(b)). In three groups, one user summarized findings by linking selective countries to map and table. Linking was also employed to clarify unknown items, *e.g.*, the exact location of a country, especially if the country's label was not visible on the map due to a coarse zoom level.

Storing and sharing findings

The bookmark list was actively used by four groups (*cf.*, Figure 9.3). However, usage of the bookmark list was usually only temporary – all but one group used the text editor to gather and structure all findings. Users complained that the bookmark list was too

restrictive. While it seemed to be sufficient to gather first findings (as also indicated by the higher usage by groups working mainly individually), users stated that crucial collaboration features, like sorting or being able to add meta-information, were missing.

Both, bookmark list and text editor, were used to "replay" previous findings by stepping through the individual items to invoke (one shot-)linking. In four groups, the list in the text editor or bookmark list – created either by a single user or by both participants – was replayed by one participant to link to the map and the table. The group then discussed and verified the individual items and decided on the most suitable candidates collaboratively. Conversely, several group members consulted the list created in a collaborative session to identify relevant locations in their articles for further reading.

Interaction with single-user applications

Although we observed that most groups could resolve input conflicts on shared application windows by social protocols, the majority of users reported that occasional input conflicts were very annoying. Conflicts were observed when panning the map or selecting text by dragging the mouse while the other user was moving the cursor across the window or when attempting to drag concurrently. Apart from input conflicts, three groups complained about having to share a single bookmark or text editor window. One user mentioned that *"sharing windows is fine as long as you are mainly viewing. Like on the map or the table. But it's exhausting if both have to interact"*. In practice, they agreed at some point to switch storage window ownership and moved it to the other side of the display or were alternatingly adding items on the original window location. This was perceived as awkward, as one user put it: *"windows on the left side of the display felt like [user B's] area"*. Obviously, users silently agreed on window ownership based on the initial window layout. This is reflected in logging data, where 96% of registered interaction with the bookmark list (*i.e.*, bookmarking or selecting a bookmark for linking) was contributed by the user sitting on the left side of the display. Similarly, 75% of linking activities from the text editor were recorded by the user sitting on the right.

Some users reported problems when accessing distant shared content. For instance, one user argued that the numbers in the table were hard to read for him. However, he *"did not dare to pull the table down as I did not want to disturb [user B]"*. Indeed, three users reported that window re-stacking or moving by the other user caused distraction for them when working individually.

9.2.7 Discussion

We will discuss the results of the experiment with regard to the research questions in this experiment.

Q1 *How does a pair of users coordinate information access and manipulation on a shared wall display using multi-input support and conventional single-user applications?*
As observed by previous work, facilitating multiple concurrent input devices and sufficient

physical space lead to an increased amount of individual work [41, 245, 246]. Indeed, we saw a lot of individual work, primarily during information retrieval phase. Users generally seemed to coordinate their activities during periods of independent work without major effort. Similar to previous observations on territoriality in single-display groupware [215, 240, 245], we saw that users mainly interacted with windows located on their physically closer side of the display without explicitly negotiating personal areas. The assignment of personal windows thereby was derived from the pre-defined window layout, which was rarely changed. Consistently with previous research [72], users reported that changes to the existing window layout were perceived as distraction and therefore avoided. The main causes of distraction mentioned by the users were occasional input conflicts when users attempted to access detailed information in a shared application window.

Window managers can support users in decreasing the amount of distractions by imposing access restrictions (*e.g.*, private screen regions [125, 244] or locked windows [115, 244]). However, such system-imposed access regulations are often considered as too restrictive [72, 90], but well-designed automatic protocols have recently been demonstrated to be preferred over pure social protocols by users [178]. In addition, systems should make it easier for users to re-arrange the window layout without distracting the other users, for instance by employing semi-transparent overlapping windows [268], duplicated window regions for closer inspection [238], or personalized views in dedicated projected channels [219].

Q2 *How does collaborative information linking support (or impede) users in information discovery and information sharing?*
The most extensive usage of collaborative information linking was for groups working in a mixed-focus collaboration style. Usually, these groups started by a joint task clarification and division step, followed by individual reading and gathering of information, and concluded with jointly discussing, clarifying, and verifying their findings. As the task was to answer three independent questions, this pattern was re-occurring. These observed collaboration patterns correspond to observations of previous research on collaborative information work (*e.g.*, [153, 176]). Individual information retrieval periods were usually supported by visual links to discover related information in shared application windows. Since previous research reported that cursor movements were perceived as distracting on a shared display [258], we were surprised to find that users hardly felt distracted by the other user's visual links. However, our experiment was limited to a pair of users. A more thorough evaluation of visual coordination techniques and the resulting visual clutter for a larger number of users is subject to future work.

We also observed that collaborative information linking reduced the need to interact with shared window content – in some situations. It seems to be useful for mere overview applications, where users are rarely required to interact – such as the map application in our experiment, where the need to zoom into the map was often mitigated. However, it could not help users to read highlighted, but small, textual labels (as in the migration table) located at a far distance. In contrast to previous observations reporting difficulties in gesturing on large displays (*e.g.*, [119, 204]), we did not observe any communication

problems when pointing to a specific display location using visual links.

In the future, it is worth investigating whether visual links can be used as interaction technique to enhance collaboration-supporting window manager techniques. As an example, users could "drag" an existing link to create a window region duplicate, like *WinCuts* [238], close to their personal interaction area.

Q3 *How does the central application coordination approach and selection storage support users in information sharing?*
Collaborative information linking and the provided storage facilities were often employed when transitioning from individual information retrieval to joint discussions. Users "replayed" selected findings gathered in the bookmark list of text editor to their partner for focused discussion and verification. Mark and Kobasa [153] discovered a similar "retracing" step when users validated answers collaboratively. Conversely, some users replayed findings discussed in the team to get an entry point for new information sources when returning to individual information retrieval.

However, the tools we provided for information storage were described as too inflexible. Users had a strong desire to properly collect and structure their findings when working in a mixed-focus collaboration style. Based on the observations in our experiment, we found that a centralized information storage tool should provide similar features as powerful application-specific interaction histories [105], but should additionally enable the users to merge multiple personalized instances and to easily invocate personalized visual highlighting to diverse applications. As the information provided in the experiment was filtered (and therefore window content was rarely changing, except for the personal browser windows), a simple selection storage mechanism was sufficient to revisit previous information occurrences. We expect that with a growing amount of information, mechanisms to restore previous application states (as discussed in Section 8.1.2) will be required.

9.3 Comparison of Techniques to Discover and Access Occluded Windows

We conducted a controlled user experiment to initially judge the usability of the *window uncovering* technique (Section 8.2) for discovering and accessing content in occluded windows. For that purpose, we compared it with two other window management techniques for three different tasks. The tasks were designed to simulate real information work situations, where users have to skim through information in occluded windows or quickly interact with obscured content before resuming the main task.

9.3.1 Research Questions

We formulated two research questions that were tested in the course of the experiment.

Q1 *Which technique supports the user best in quickly looking at information in occluded windows?*

We expect uncovering windows to be advantageous for discovering information in occluded windows, as the number of necessary steps to reveal occluded content is reduced.

Q2 *Which technique supports the user best in simple interaction tasks in occluded windows?*
We expect a performance benefit of uncovering windows for simple access to information in occluded windows, as we provide the facility to directly interact with user interface elements in occluded windows. In this way, information access and discovery are no longer two separated activities.

9.3.2 Participants

We recruited 15 participants (4 female, aged 15 to 32) from a local high school and university. All participants were experienced computer users and tested for color-blindness. Nine participants are using Microsoft Windows as a primary operating system, three employ Linux and three use Mac OS X. Twelve of the participants use Alt+Tab "often" to "very often" to switch windows, followed by the window list in the task bar, which is employed frequently by eleven participants. None of the participants stated using one window switching technique exclusively.

9.3.3 Window Management Techniques

We compared the following three window management techniques for revealing occluded window content:

(a) (b) (c)

Figure 9.5: The three window management techniques and the three task types in our experiment: (a) overlapping windows with the Alt+Tab menu for the interact task, (b) free-space transparency [123] with the count task, and (c) importance-driven compositing (Section 8.2) with the read task.

Alt+tab (AT) in combination with conventional overlapping windows is a standard window switching technique provided by all major operating systems and served as a control condition. The employed Alt+Tab switcher by the Compiz window manager shows small previews of all windows when activated by the Alt+Tab sequence (Figure 9.5(a)), and allows for sequential window selection by pressing the tab key.

Free-space transparency (FST), proposed by Ishak and Feiner [123], applies transparency to unimportant window regions of overlay windows using a smooth gradient between transparent and opaque regions. We simulated FST using importance-driven compositing without the layout routine (Figure 9.5(b)). Instead of the original notion of unimportant window content (white pixels), we used our importance maps to define transparency values. FST does not allow users to directly interact with the occluded content. Therefore, we provided the "pie menu" proposed by the authors, which shows a circular menu of all the windows lying underneath the current pointer location, to bring occluded windows to the front. Participants had to press Start+tab to retrieve this menu and then click on a window preview to bring the desired window to the front.

Importance-driven compositing (IC) was employed to uncover occluded windows on demand, as described in Section 8.2. To initiate IC, participants had to press the key combination Start+w. As long as the keys were pressed, occluded windows were spatially arranged, cut-aways were applied and interactivity for the most salient window underneath the pointer was ensured (Figure 9.5(c)).

9.3.4 Setup

The study was conducted on a PC running Linux Ubuntu 10.04 with a 1280x1024 17" monitor. Users operated the PC with a conventional mouse and keyboard pair.

9.3.5 Task

Participants were asked to solve variants of a visual search task on application windows. The task involved a series of questions in textual format. The question was presented in a maximized main window in the foreground. Users were asked to identify specific items in five small object windows, occluded behind the main window. The object windows were arranged in a cascaded fashion – a typical window layout strategy of conventional window managers. The object windows always contained an image of a 2D geometric primitive and – depending on the task type – a small textual label and a button. In the main window, users were presented with a list of answers below the question (in two task types), where the user had to select the correct answer. In one task type, the correct answer had to be directly selected in one of the object windows.

The following three task types had to be solved:

Count: Participants were asked to look for a certain 2D geometric primitive (*e.g.*, square) in the five object windows, count its occurrences, and select the number of occurrences from the list of solutions in the main window (*cf.*, Figure 9.5(b)). This represents a scenario where users have to get an overview of all the windows and sequentially scan them [135] for a strong visual feature, which is also clearly visible in a scaled window representation.

Read: Participants had to find the only object window containing a textual label (*e.g.*, *"This is a yellow triangle"*) that matched the associated picture of a geometric

primitive (*cf.*, Figure 9.5(c)). Subsequently, they had to select the same label from the list in the main window. This represents a scenario where the user has to switch to a window containing information required for the main task. Note that the text labels were too small to be readable in the FST pie menu and AT preview menu.

Interact: The task was similar to the read task, except that the validation of the matching label was selected by a push button directly in the object window (*cf.*, Figure 9.5(a)). The task was designed to represent situations where the user has to shortly interact with an occluded window before resuming the main task.

The tasks were very simplistic on purpose, so no prior knowledge was required and users could solve the task without major mental effort. In this way, differences in task performance can be directly linked to the employed user interface. Bly and Rosenberg [43] employed a similar task when comparing a tiled window manager with overlapping windows: Users had to match graphics objects with brief paragraphs describing them.

Our initial window layout was chosen to simulate a display cluttered with windows. As our primary goal was to assess the discovery and access of information in occluded information, we employed only a single fully visible window, obscuring information in multiple secondary windows.

9.3.6 Design and Procedure

The study followed a 3x3 within-subjects factorial design with the following factors:
technique: AT, FST, and IC, and
task: count, read, and interact.

We measured the completion times (time between appearance of a question and selecting an answer) and error rates for each task item. Participants were also handed out a preference questionnaire at the end of the experiment. A semi-structured interview was conducted to collect subjective feedback. For each technique, participants had a short practice session (6 questions). The order of the techniques and tasks was counter-balanced. Each user had to complete four repetitions for each task in every technique. Results from these repetitions were accumulated. For the interact task and read task, the stacking position of the window containing the correct label was balanced across the repetitions.

9.3.7 Results

We conducted a 3 (Technique: AT, FST, IC) x 3 (Task: Count, Read, Interact) repeated measures ANOVA ($\alpha = .05$) to evaluate completion times. Bonferroni adjustments were applied for post-hoc comparisons. We found a main effect for Technique ($F_{2,28} = 82.231, p < .001$) and Task ($F_{2,28} = 18.182, p < .001$), as well as a borderline significant interaction between the two factors ($F_{4,56} = 2.601, p = .046$). Post-hoc comparisons showed that IC (6.2s) was significantly faster than both, FST and AT (13.9s and 8.4s). AT was also faster than FST. However, IC was only faster than AT for the read and interact tasks. For the count task, there is no significant difference between IC

and AT, but both techniques were performing better than FST. Figure 9.6 illustrates the completion time results.

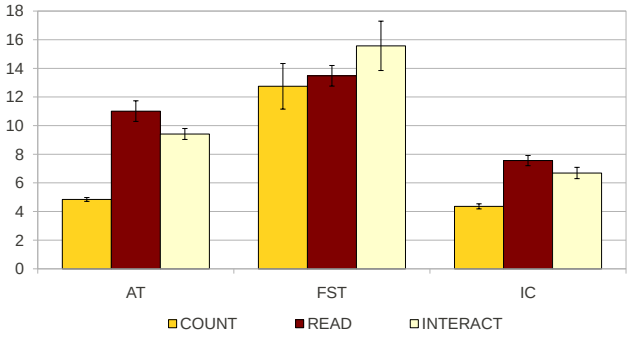


Figure 9.6: Completion times (seconds) for the three conditions (AT, FST, and IC).

Participants generally committed few errors with 97% of the questions being answered correctly. The highest error rates were collected for FST in the count and read tasks (10.0% and 6.7%, respectively).

Participants were asked to rank the three techniques on a seven-point Likert scale. A Friedman non-parametric test revealed a significant difference between user preferences ($\chi^2(2) = 23.414, p < .001$). Post-hoc comparisons using Wilcoxon Signed Rank tests with Bonferroni adjustments showed that IC (6.33) was rated significantly higher than AT (4.73), and that both techniques were evaluated higher than FST (2.47). Preference scores are illustrated in Figure 9.7.

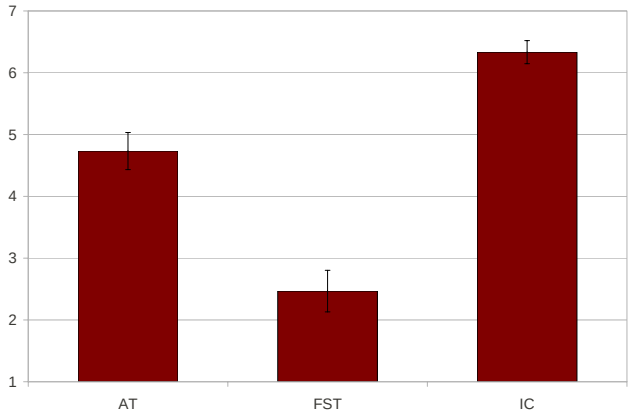


Figure 9.7: Preference scores for the three conditions (AT, FST, and IC) on a 7-point Likert scale.

In the interview, participants mentioned the readability of small text and the ability to interact with obscured user interface elements as main reasons to rank IC higher than FST and AT. FST was primarily disliked for the pie menu to access occluded windows, which was described as hard to use, because only the windows located underneath the pointer were shown. Due to the initial cascaded window layout, only the top-most object window

was fully visible (*cf.*, Figure 9.5b). Therefore, several users commented that they did not know which windows were located beneath the mouse pointer and, as a consequence, which windows were included in the menu. One user summarized interaction with FST as: *"Transparency and not being able to interact [with occluded content] is very exhausting"*.

We also asked the participants which technique they preferred for the three tasks. For the interact task and read task, IC was chosen by the majority (15 and 12 participants out of 15, respectively). In the interview, most participants stated that IC was most appropriate for the read task as the text in obscured windows was readable, in contrast to the small menus of AT or FST. For the interact task, participants mentioned the ability to directly interact with occluded content without explicitly selecting the corresponding window as exceptionally useful. For the count task, AT was the most preferred technique (selected by 8). Participants commented that the menu of AT provided a good overview, so they could immediately see which images were available.

9.3.8 Discussion

The results of our experiment indicate that importance-driven compositing supports users in skimming through information in occluded application windows, thereby helping them to discover and access occluded information. We will discuss the results and implications with respect to the research questions.

Q1 *Which technique supports the user best in quickly looking at information in occluded windows?*
The uncovering windows technique using importance-driven compositing was shown to indeed provide an advantage for discovering content in occluded windows. However, compared to conventional Alt+tab window switching, the performance advantage is only significant for accessing fine-grained information. This is expressed in the superior performance of users in the read task for IC, while Alt+tab had a comparable performance when looking at coarse and easily identifiable window content. Most participants appreciated that they could read the text in the obscured windows, in contrast to the small menu of Alt+Tab or the pie menu of free-space transparency. In these cases, they first had to switch to the anticipated object window to read the text and then back to the main window to select the correct answer. Two participants therefore described our technique as *"more fluid"*, as only one activity was necessary to extract information from a hidden window.

Q2 *Which technique supports the user best in simple interaction tasks in occluded windows?*
The benefit of the uncovering windows technique for simple interaction in occluded windows was confirmed by the results of the interact task. The ability to directly interact with content of occluded windows without explicitly selecting the window in a separated user interface is a distinguishing aspect of our technique compared to other window switching interfaces. Being able to see occluded content without the ability to access the information with the pointer using the free-space transparency technique was strongly criticized by our

users.

There are several interface aspects that have not been captured in the course of this experiment. For instance, as windows were re-loaded after each question, users were not required to re-visit previous windows. However, window re-visitation is a frequent activity in conventional desktop computing [236]. Tak *et al.* [236] showed that spatial stability of the window switching interface is a core requirement for quick re-visitation. As importance-driven compositing prioritizes windows in the layout routine based on their stacking order, windows are possibly uncovered in a different spatial layout when re-initializing after changing the window stacking order. One user noticed the spatial instability of uncovered windows even in our experiment, where re-visitation was not required to solve the task.

In addition, we cannot fully assess the effect of the dense information display of our uncovered window interface on focused attention, as the number of application windows was limited. One user expressed a certain dislike for the resulting visual clutter and requested a more "tidy" window arrangement with neatly arranged window boundaries, similar to the well-known Exposé interface.

Alignment of window boundaries and displacement from the previously calculated window location could be added as additional penalties of the window layout optimization problem of importance-driven compositing (*cf.*, Section 7.4) to address these user requests.

9.4 Exploratory Evaluation of Window Management on Irregular Displays

To initially evaluate selective features of display-adaptive window management (Section 8.3) and to assess the future directions for window management on irregular displays, we conducted an exploratory experiment. On an irregularly shaped display, users were asked to solve an information analysis task involving multiple application windows with two window management techniques: conventional window management without spatial awareness and window management with selected display-adaptivity features.

9.4.1 Research Questions

The aim of the experiment was to assess three research questions:

Q1 *How do users manage their windows on an irregularly shaped, non-planar display?* Although window management strategies have been explored for very large displays [4, 32] and multi-monitor settings [91, 112, 113], there has been so far been no attempt to observe window management behaviors on irregularly shaped displays – to our knowledge. Indeed, it might seem inappropriate to use interaction and presentation techniques designed for usage on limited screen space with clearly defined, strictly rectangular outlines on an irregular display. However, we intent to informally observe and question users to initially assess the basic suitability of such displays for conventional window management, and to furthermore discover emerging interaction patterns to cope with the situation.

Q2 *Does display-geometry snapping support users in managing windows?*
Previous research has shown that users attempt to partition their workspace, for instance by explicitly exploiting multiple discontinuous monitors [91]. Similarly, we hypothesize that users will facilitate separations afforded by physical discontinuities, such as corners, on projected displays. We furthermore hypothesize that our display-geometry snapping technique (Section 8.3.1) supports users in exploiting these separations. In addition, we speculate that snapping will help users to quickly arrange windows within the usable display regions – especially if the outline of the projected display is not visible.

Q3 *Does semi-automatic window coordination support users in managing windows?*
For large-scale displays, it has been demonstrated that window management overhead for the user increases – especially for basic operations, like moving and resizing windows [32]. This management overhead is caused by an increased number of open application windows [112], the attempt to make most of this information visible and directly selectable [113], as well as an increased amount of spatial organization [4, 32]. Thus, we hypothesize that automating some of these activities by providing semi-automatic window coordination (Section 8.3.2) will decrease the amount of manual window management operations and thereby increase the user's performance in an information analysis task.

9.4.2 Participants

We recruited 8 experienced computer users from a local institute (1 female, aged 27 to 33). Five users were primarily using Microsoft Windows as operating system, two Linux, and one Mac OS X. All users stated to work with a dual-monitor setup on a regular basis, where the size of the larger monitor was given as 22" by one user and 24" for the others. Despite this considerable amount of available pixels, half of the participants indicated to usually maximize their windows to the monitors.

9.4.3 Window Management Techniques

To accomplish the information analysis task, we limited the set of available window interaction techniques to a single activity: window moving. Resizing, maximizing, minimizing, or closing of windows was not supported, so we could evaluate this single aspect in full detail. Also, we did not support any window switching techniques like Alt+tab or the Exposé. We solely allowed users to move windows by dragging them along the title bar.

Users had to accomplish the task with two window management techniques:

Manual window management (M) was limited to the ability to drag a window and to change the stacking order by directly clicking within the window's boundary.

Display-adaptive window management (DA) additionally snapped the windows to be entirely contained within a planar, fully visible display area, if dragged outside or if the dragged window was released on top of a physical corner. In addition, it supported semi-automatic window coordination on demand. In a pilot study, we discovered that constant semi-automatic coordination was perceived as too "patronizing". Therefore, we

provided it as an optional feature: if pressing the Start key while releasing a dragged window, underlying windows were subject to semi-automatic coordination. Conventional dragging of windows did not influence the spatial window layout of underlying windows.

For both techniques, we enabled warping and blending to compensate for projection discontinuities.

9.4.4 Setup

The experiment was conducted on an irregular, tiled display driven by two XGA projectors, connected to a PC running Ubuntu 10.04. The display spanned two planar display regions, where the left region was larger and also contained the overlap region between the two projectors. Due to the strongly oblique projection angles, some parts of the display suffered from noticeable interpolation artifacts when applying warping and blending – in particular the right half of the left display area. The user was sitting on a table facing the left display area, approximately centered. The windows were controlled by a conventional mouse and keyboard pair. Figure 9.8 shows the employed display setup.

(a) (b)

Figure 9.8: Irregular display setup for evaluation of display-adaptive window management: (a) the two projectors (on the left) driving the irregularly shaped display and (b) the resulting display with invisible outlines during the study.

Clearly, this setup is not considered as an optimal solution, especially due to the interpolation artifacts. However, our aim was to simulate office environments where space restrictions often require non-optimal projector setups, resulting in similar irregularities. Also, it would have been desirable to increase the resolution by using more projectors and keeping the size of the display approximately constant. As we used very small application windows in our task (Figure 9.8(b)), we could simulate this effect even on a low-resolution display.

9.4.5 Task

Users were presented with eight to ten small content windows containing a picture of a car, its name, and other attributes, such as price, power, and mileage. Additionally, a sightly larger main window contained instructions for the task. For each task, a new set of windows was loaded – initially in a cascaded arrangement, with the main window placed on top of the cascaded window stack. Using these windows, the users had to solve three task types:

Sort: Users were asked to sort the content windows according to some attribute (*e.g.*, the price of the cars) in a linear sequence. We did not explicitly instruct the users how the resulting arrangement should look like. They were only told to make the ordering clearly visible and understandable. As this task was assessed purely visually by the study supervisor, we did not quantitatively evaluate the correctness. This task should represent task environment setup activities [135].

Count: To simulate a sequential scanning task [135], we asked our users to count all cars of a given brand. Thus, they had to sequentially visit all windows and count the number of occurrences. The number then had to be entered in the main window.

Compare: As a complex comparison task [135], we asked the users to find the most suitable car by evaluating three given parameters (*e.g.*, the cheapest car with at least 10 km/l mileage and at least 5 seats). Thus, they had to sequentially scan all content windows to filter those cars violating the given constraints. Subsequently, they had to scan the remaining windows for the most appropriate parameters. The most suitable candidate then had to be selected by directly clicking a button in the content window.

9.4.6 Procedure

For each window management technique, users had a warm-up period with one repetition for each task type. Users were encouraged to "think aloud", to explain their window management strategies, to mention what they liked and what they disliked about the physical environment or the current interaction technique. Subsequently, the users had to accomplish two repetitions of each task type in an actual trial, where we logged task completion times and correctness. Additionally, the experimenter took notes during the thinking-aloud warm-up period, as well as during the actual trials. After each run, users had to fill out a questionnaire. After both runs had been accomplished, users were asked to assess the two window management techniques, indicate how much they used the display-adaptivity features, and how they liked the display setup overall. A semi-structured interview was conducted at the end of the experiment. Overall, participation lasted approximately one hour.

The sequence of window management techniques was counter-balanced across the participants. Additionally, the task sequence and the initial window stacking order was randomized.

As we collected a considerable amount of observations and user feedback, we issued the

users a follow-up questionnaire after the experiment. The questionnaire contained informal observations and statements by users about the display arrangement, their employed window management strategies on the irregular display, and suggestions how to improve window management in such an environment. Users were asked to indicate how much they agree with the statements. From the initial eight participants, seven users returned the questionnaire, as well as one pilot user. We used these *agreement* values for evaluation of research question Q1, concerning general window management strategies on irregular displays. Also, suggestions for window management techniques by our participants can be seen as indications for future research directions.

9.4.7 Results

We will discuss the results of this experiment by presenting observational evidence and informal user feedback. In addition, we will report on task completion times (measured in ms), questionnaire results (7-point Likert scale), and results from the follow-up questionnaire, expressing the agreement of the users with certain observations, feedback, and suggestions for future directions (7-point Likert scale), summarized in Table 9.1.

Display Arrangement and Emerging Window Management Strategies

In the manual window management condition – in particular in the thinking-aloud warm-up period – we observed emerging window management strategies of our users to deal with the irregularities of our display setup. Overall, users assessed the display arrangement with 4.25 on a 7-point Likert scale. They mentioned several reasons why they would *not* want to have such a display in their offices, where the most agreed point of criticism was the blur in some areas, which lead to readability problems. Interestingly, the physical corner was judged as rather useful by most participants, and some users mentioned they *"explicitly used"* the physical separation to create meaningful spatial window arrangements. Although we observed that users occasionally placed their windows across the physical corner (for six participants), users largely denied that they did not care about the window placement with respect to the physical corner. Users mentioned the reduced readability and *"awkward interaction"* as reasons not to span windows across the corner. Also, the irregular display outline was not particularly disliked, although participants largely agreed that having the outline visible would have been an advantage. However, users did not invest much effort to keep windows within the visible display area. One user explained it as: *"I kept the important windows in the inner part of the display, and for the others, I did not care if they were cropped".*

All users employed some sort of "piling" to manage their windows – a well-known strategy of knowledge workers with physical paper [149, 150] and with windows on large-scale displays [4]. The number and spatial arrangement of piles differed among participants and task types. The most common type of pile was a "dump" pile, where irrelevant windows were casually stacked on top of each other. Four out of the eight participants established

Display arrangement		
I disliked the separation of the display caused by the physical corner.	1.88	0.48
I disliked that the display was not rectangular.	3.50	0.78
I disliked that the display outline was not visible.	5.13	0.64
I disliked that some parts of the display were blurry and therefore hard to read.	6.25	0.31
I disliked the low resolution.	3.63	0.63
I disliked that the display was so large.	2.00	0.50
I disliked that the display was so high.	3.50	0.73
I disliked that I was sitting so far away from the display.	2.50	0.38
Window management strategies		
I used the right area as "deposit area" or (recoverable) "trash can".	6.00	0.60
I used the right display area to show persistent background information.	4.38	0.86
I used the right area for both: depositing items and displaying background information.	5.25	0.73
I did not have a clear strategy how to use the two display areas.	1.88	0.40
I would not have used the right display area if I had had the option to close or minimize windows.	1.88	0.40
I did not care whether windows were spanning across the physical corner.	2.38	0.63
I did not care whether windows were partially located outside the visible display outline.	3.88	0.74
I never moved my windows so far outside that they were cropped by the display outline.	3.38	0.82
Window management techniques for "context areas".		
I would like to "throw" windows to distant display areas with a short mouse gesture.	6.13	0.30
I would like to have a button in my window title bar to move my window to a distant display area.	3.00	0.57
I would like windows to be smaller (scaled or somehow cropped) at distant areas.	4.00	0.65
Windows located at distant display areas should be easily recognizable.	5.88	0.44
Windows located at distant display areas should be easy to move back to a closer area.	6.38	0.26
Information on distant windows has to be readable.	4.38	0.56
Window management techniques for "focus areas".		
I would like an "explode" feature to arrange my windows in a (messy) non-overlapping fashion.	4.63	0.46
I would like a "grid" feature to arrange my windows in a regular grid.	4.63	0.73
I would like a "snap" feature to quickly align windows next to each other.	6.00	0.38
I would like to manually group windows into common containers and move them together.	5.88	0.35
I would like to move an existing "pile" of windows together.	5.75	0.45
I would like to have only one maximized window on each display area.	3.00	0.46
I like to have empty space available (e.g. as a "buffer zone").	5.63	0.46

Table 9.1: Results from the follow-up questionnaire: the recorded statement and how much the users agreed with this statement (mean and standard error on a 7-point Likert scale).

their dump pile on the right display area, as illustrated in Figure 9.9(a). Windows identified as irrelevant were quickly dragged to the right display half – usually without even looking. Two users created dump piles by dragging windows partially outside the visible display outline (Figure 9.9(b)). Two participants employed both strategies, depending on the task type. Three users also created piles for "usable" candidate windows – especially for the compare task, where windows had to be visited multiple times. In contrast to the dump piles, these piles were located centered on the left display half (Figure 9.9(b)). As these piles were usually re-visited, the established pile structure was subsequently resolved, so information could be compared side-by-side with reduced occlusion. Others arranged usable candidate windows in a linear row (Figure 9.9(a)), in a regular grid, or kept a single, most suitable candidate in a prominent location and placed one window after the other next to it for pair-wise comparison.

Users agreed that they established an explicit strategy how to facilitate the right display area. Apart from using the area to pile up irrelevant windows, users also reported

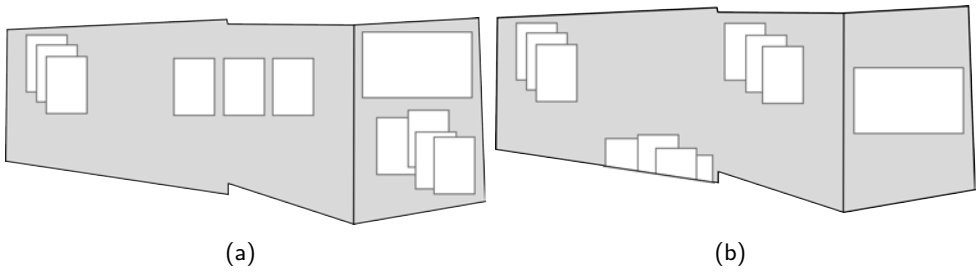


Figure 9.9: Exemplary window piling strategies observed during the experiment, illustrated on the 2D display importance map of the employed display setup: (a) from the original, cascaded stack on the left, useful candidates were linearly aligned in the focus region. The main window was carefully placed in the context region, next to casually piled up irrelevant windows. (b) Others arranged relevant candidates in a pile and placed irrelevant windows partially outside the visible display outline on purpose.

that they facilitated the area for placing persistent background information which did not require any further intervention, such as the main window. This information was carefully positioned and was kept uncovered during the entire task, so they could quickly access the information by just turning their head. Interestingly, most users agreed that the right area would have been useful even if the option to minimize or close windows had been available. One user explained this opinion by *"It would have cost more time to click the close button. Moving the window to the right was faster."* Another user mentioned the ability to quickly re-acquire the information as more appropriate compared to closing the window. This corresponds to the observed tendency to keep more windows open if sufficient display space is available [112].

It has to be noted that our windows were rather small and the number of pixels was low in our setup. In fact, the overall number of pixels (2560x1024) was lower than in all of our participants' everyday work environment. Assuming a similar control/display gain of the mouse, the display could be traversed with comparably little mouse movement. Still, some users mentioned that dragging the windows to the right display area caused a lot of effort *"because the display is so large"* and suggested simple mouse gestures or buttons in the window title bar to quickly relocate windows to distant areas. Especially the suggestions of having a "throw" gesture (*e.g.*, as proposed by Geißler [88]) was well received by the users (Table 9.1). Also, having the possibility to move multiple windows concurrently (either manually grouped or existing piles) was suggested to decrease window management operations (*cf.*, *snapping windows* [25] or *storage bins* [213]).

Display-Adaptive Window Management

Contrary to our expectations, our display-adaptivity features could not enhance the users' performance. Task completion times for the three task types were almost equal, with display-adaptive window management being even slightly slower than manual window

management (sort: $t_7 = -.460, p = .660$; count: $t_7 = -1.611, p = .151$; compare: $t_7 = -.172, p = .868$; using paired t-tests). Correctness values were not evaluated separately, as only three questions were answered incorrectly in sum (all in the compare tasks).

However, we found some differences in subjective assessments by evaluating the questionnaires using Wilcoxon Signed-Rank tests. Managing the windows in general was assessed as easier with display-adaptive window management compared to manual window management ($Z = -2.11, p = .035$). Users indicated that they had a better overview ($Z = -2.060, p = .039$) and that they could more easily sort ($Z = -2.047, p = .041$) and browse ($Z = -2.232, p = .026$) their windows. Comparison across multiple windows ($Z = -1.414, p = 1.57$) and re-visitation of windows ($Z = -1.134, p = .257$) was assessed as equally demanding. Users also indicated that they spent more time moving windows around in the manual condition ($Z = -1.983, p = .047$). However, the time spent looking for windows was assessed similarly ($Z = -1.000, p = .317$). Questionnaire results are visualized in Figure 9.10.

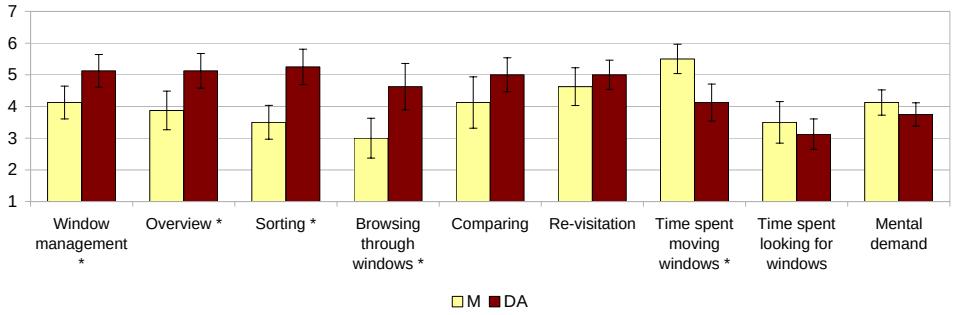


Figure 9.10: Questionnaire results for comparison of manual window management (M) and display-adaptive window management (DA) on an irregular display (mean and standard error on a 7-point Likert scale). Significances ($p < .05$) are indicated by *.

Overall, users ranked display-adaptive window management slightly higher than manual window management ($r_{da} = 5.0$ and $r_m = 3.375$), which is borderline significant ($Z = -1.930, p = .054$).

User feedback indicates that display-adaptivity subjectively supported users in the sort task, *i.e.*, when setting up their environment. Also, the increased amount of subjective overview and productivity gain while browsing the windows may have helped users during task execution – both sequential scanning (represented by the count task) and complex comparisons. Indeed, display-adaptive was chosen as the more suitable technique for the sort task (87.5%) and for the compare task (62.5%), while preferences for the count task were balanced.

In the post-study questionnaire, users had to indicate whether they explicitly employed display snapping or semi-automatic window layout (from “not at all” to “very often” on a 7-point Likert scale). Display snapping usage was rated low with 2.63 on average, while subjective usage of semi-automatic window layout was higher with 5.38. Indeed, we also

observed that display snapping was facilitated rarely. Most users never placed windows close to the display boundary and if so, only irrelevant windows where the placement was not considered as important. For some participants, display snapping even interfered with their intentions. Two users mentioned that they intentionally wanted to place windows partially outside the visible region to create a "dump pile" (Figure 9.9(b)), but the system would automatically move their windows back in. As a result, valuable display space was covered with irrelevant information. No user complained in the interview that windows could not be spanned across the corner – even though we observed occasional window placements around the corner in the manual condition for six users. In contrary, one user claimed that *"snapping at the edge is important"*. Another user explained that edge snapping would be *"even more important if the displays were slightly discontinuous"*. One user mentioned that display-geometry snapping would be useful in combination with a "throwing" gesture [88].

The semi-automatic window layout feature was facilitated by most users. We observed three usage types: Users employed it as an "explode" tool to resolve the initial cascading window arrangement at the beginning of the task. Users commented that this was a fast way to get a quick overview. However, one user demanded more control about the "force" of the tool and another one wanted it to be generally *"more aggressive"*.

The second usage type for semi-automatic window layout was to "squeeze in" windows into an existing spatial layout – usually towards the end of the task. This was a common approach to solve the sort task.

Finally, some users facilitated semi-automatic window layout to "explode" manually created piles in the middle of the task. Usually, they did some coarse presorting (Figure 9.9(b)), followed by an explosion of the pile. Subsequently, they visually scanned the individual windows for a more detailed comparison. Thereby, some users expressed the wish for a more "tidy" arrangement, such as a regular grid, for easier visual comparison.

User feedback also helped to identify conceptual problems of existing display-adaptive window management features. Display snapping was sometimes considered as unintuitive as it aimed to find the closest location of a dropped window within the visible display boundary with respect to the drop location, while not taking the movement direction into account. As the display outline was not rectangular, the window did not only snap vertically and horizontally, but according to the angle of the (invisible) boundary. Users also complained that the semi-automatic layout feature sometimes led to an unintuitive window layout or destroyed a previously established spatial layout. Indeed, we observed that our initial approach to prioritize windows for the layout algorithm according to their stacking order (*i.e.*, their recency of use), as described in Section 7.4, may lead to unpredictable results on a large display. A more appropriate priority queue would lay out the windows according to their proximity to the drag window, so windows gradually move away from the drag window.

9.4.8 Discussion

Results of our experiment show that users established unconventional window management strategies to cope with the size and irregularities of the display. As diverse patterns emerged among the participants, our display-adaptivity features did not support all the users as much as initially expected. However, from our observations and user feedback, we could identify user interface problems of our features and establish future research directions to support users in their emerging window management strategies.

Research Questions Revisited

Q1 *How do users manage their windows on an irregularly shaped, non-planar display?*
Irregular display are not the most obvious choice for conventional information work. A common goal in related research therefore has been to visually compensate for irregularities in projected displays (*e.g.*, [40, 169, 189]). However, our exploration indicates that users do not necessarily dislike certain irregularities, such as physical corners or non-rectangular projection outlines. In contrast, they explicitly facilitated physical discontinuities to separate their workspace into *focus areas*, where their primary windows were located and most interaction took place, and *context areas*, where irrelevant windows were casually piled up and persistent background information was placed.

Q2 *Does display-geometry snapping support users in managing windows?*
Contrary to our expectations, users rarely made use of the display snapping facility. We observed that most windows were never located close to any display boundary. If windows were dragged close to the border – or partially outside – they were usually not relevant, so users either did not care about their placement, or intentionally wanted to make them partially disappear to decrease the amount of occupied display space. In the latter case, display-geometry snapping was even perceived as annoying rather than useful. In contrast, users commented positively on snapping to the physical edge.

According to these observations and informal user feedback, we found several opportunities to improve display-geometry snapping. In the future, snapping will be less aggressive at the boundaries compared to physical edges. This can easily be achieved by applying lower importance values to the cropped areas in the display importance maps. In addition, snapping was currently considering windows to be opaque and thus moved them back into the visible display region entirely – irrespective of the window content. Content-awareness may allow the user the hide irrelevant window areas along the boundary while keeping important portions visible for easy recognition. As described in Section 8.3.1, this feature is available, but was disabled for the experiment.

Q3 *Does semi-automatic window coordination support users in managing windows?*
In contrast to display-geometry snapping, semi-automatic window layout was frequently employed by most participants – mainly to "explode" piled up windows or to "squeeze in" windows into an existing spatial layout without manually adjusting the remaining windows. While the main purpose of the "explosion" of existing piles was to increase visibility for

visual comparison and *"to get a better overview"*, users disagreed about their anticipated exploded window layout: about half of the user tended towards a messy layout, while the other half clearly preferred a regular grid layout. As piling is a well-known strategy for management of paper on the desk [149, 150] or windows on large displays [4], interaction techniques to resolve piled documents on desktops have been researched and implemented in the past [1, 150].

In the future, we require a better prioritization criterion than the currently used reversed window stacking order to improve the quality of the resulting window layout. Sorting the windows according to their proximity to the drag window for the layout algorithm (described in Section 7.4) seems to be a promising measure. Furthermore, users suggested to show a preview of the resulting window layout and the opportunity to interactively control the "force" of the semi-automatic coordination – in other words: the penalty of the window displacement on the layout algorithm.

Future Directions

We observed a clear tendency of the participants to divide the display into a close focus area and a distant context area. In contrast to planar displays, where the transition from focus to context is rather blurred [32], users facilitated the physical corner as hard boundary.

When asked, some users indicated that windows in context areas should be smaller (*e.g.*, showing only a thumbnail or cropped version of the window). The most important properties of windows in context areas are that they can be easily moved back to the focus area and they are easily recognizable, according to our users. Content readability was judged as less important. The concept of scaled-down [201] or cropped [114, 155] context windows has already been suggested in previous work for planar large displays and could be similarly employed for context areas of irregular displays.

According to user feedback, we identified two types of windows in context areas of irregular displays: First, *background windows* hold valuable context information, which requires frequent visual access but little interaction. Background windows are conventionally dragged into a context region and manually positioned by the user. Important information should remain uncovered by adjusting the existing windows' positions, if possible, so quick visual scanning is supported.

Second, *dump windows* are (temporarily) irrelevant windows, which are moved to a distant location to keep valuable focus areas unoccupied. As irregular displays obviously increase the subjectively perceived mouse navigation effort, simple mouse gestures (like *throwing* [88]) should support the user in easily relocating the window to the context area. After the throw-gesture, windows should snap to a suitable region – neither occluding any background information window, nor spanning across a physical corner or cropped display outline. Dump windows should be scaled down so recognizability is supported, while keeping the amount of occupied space to a minimum. This concept is similar to *montages*

in the *Kimuara* system [143], where windows of suspended tasks are scaled down, grouped into piles, and presented on a peripheral display.

In order for our display-adaptive window management concept to better support the user in dealing with irregular displays, we need to incorporate an increased amount of high-level information. Physical discontinuities – either introduced by irregularities in projected displays or by employing discontinuous displays – need to be analyzed with respect to the users' seating arrangements, so they can be automatically segmented into physically close focus and more distant or less conveniently oriented context areas. Interaction techniques, like the variant of throwing [88], described above, and multi-window operations, support the user in quickly moving windows between these focus and context regions. Display-adaptive window management based on importance-driven compositing, as described in this thesis, is then responsible to locally adjust the window layout according to some given high-level information. For instance, it ensures that windows do not span physical discontinuities, do not cover background windows, and acts as foundation to explode window piles on demand.

9.5 Discussion

In two exploratory studies (Section 9.1 and Section 9.2), we observed that users could efficiently *discover* related pieces of information contained in multiple application windows, using visual links across applications to visually filter a large amount of visible information. In addition, we could show a benefit of importance-driven window compositing for discovering content in occluded windows (Section 9.3).

We could furthermore show a benefit of importance-driven window compositing to quickly *access* information contained in occluded application windows by either facilitating a dedicated uncovering technique (Section 9.3) or by automatically resolving occlusions of piled up windows (Section 9.4). In contrast to more conventional window switching techniques, our window uncovering technique combines information discovery and access by not only visually revealing occluded information, but also letting the users interact with this revealed content. This feature was very much appreciated by our users, who mostly disliked the free-space transparency technique [123] because of its disability to interact with occluded content, albeit being visible.

In contrast, the lack of support for convenient information access was a major drawback of visual links, especially when working on a large, shared display. We observed that users had difficulties accessing distant content highlighted by visual links, as any window layout change would lead to distraction of the team partner.

Our exploratory study on display-adaptive window management (Section 9.4) showed that irregular displays do not constrain the user in information *manipulation* by spatially arranging individual windows on the screen. In contrary, users explicitly facilitated physical discontinuities for spatially laying out windows when solving a complex information analysis task. On-demand semi-automatic window layout subjectively supported the

users in establishing a meaningful spatial layout for visual comparisons (by "exploding" manually piled up windows) and for setting up their task environment (by automatically "squeezing in" windows into an existing layout). However, we discovered that even more powerful window management techniques – aware of both, the environment and the window content – were expected, to adequately support information manipulation activities by the user.

With respect to information *sharing*, we observed that users working in a mixed-focus collaboration style facilitated central storage facilities and collaborative information linking to individually collect findings and to subsequently "replay" these findings to their partner (Section 9.1). However, central storage facilities should be much more sophisticated, as we observed that users have a strong desire to properly prepare and structure their findings before discussing with their partner.

From these first experiences, we can formulate some implications for designers of future window managers of emerging display environments:

Provide users with combined information discovery and access.
We observed that users appreciate the combination of a visualization technique to discover information with an interaction technique to quickly access the discovered item. In the future, it will be worth investigating novel window manager techniques combining information discovery and access for large displays or multi-display environments. Off-screen visualization techniques may lead the user's gaze to distant display locations outside the field of view and can be combined with interaction techniques to quickly teleport the pointer to the respective location (similar to *hopping* [118] on small displays) or, conversely, to move the remote object (or a copy of the object, similar to *WinCuts* [238] or *drag-and-pop* [17]) closer to the user.

Provide users with unified cross-application visualization and interaction.
User feedback for visual links across applications indicates that users appreciate a consistent visualization technique across multiple applications. At the same time, they also expect a consistent interaction technique to initiate or furthermore manipulate the visualization on the desktop. A central application coordination approach as proposed in Section 7.2.1 therefore needs to support bidirectional communication: It is not sufficient that applications report selections and selection occurrences. Instead, user selections should be captured centrally in a consistent manner – for instance by the window manager – and only content parsing should be conducted by the applications themselves.

Provide users with display discontinuities to enable meaningful spatial window arrangements.
Creating spatial window arrangements can be considered as *schematizing* [4, 182] to support sensemaking tasks. As shown by previous research, users facilitate multiple monitors for partitioning their information space [91]. We could demonstrate that this observation is also true for irregular projected displays, where non-rectangular outlines and physical discontinuities restrict the user in their potential spatial arrangements. Users explicitly facilitated these irregularities to spatially separate currently important focus windows from

temporarily irrelevant windows, or background windows providing persistent context information. Our provided display-adaptivity features (Section 8.3) supported users in some situations but were probably not powerful enough. We speculate that increased spatial awareness and content awareness of the window manager can support the user even better in these spatial information organization tasks.

Provide users with sensemaking tools to collect and share information from multiple sources.
Today's window managers mainly support users in information foraging, while leaving sensemaking to more specialized applications, such as visualization software. Due to the lack of a central application coordination mechanism, certain stages of the sensemaking loop (Figure 1.5) cannot be supported in WIMP environments, such as "shoebox", "evidence file", "schema", or "presentation". For our collaborative information analysis experiment, we provided users with a very simple bookmarking tool to quickly gather findings from different sources in a consistent format. Unfortunately, this tool turned out to be too inflexible. Bookmarked elements were too abstracted to be considered as a "shoebox" or as a "schema", which is why users ended up carefully structuring text files – from an initial, individual "shoebox", to an individual, more filtered "evidence file", to a collaboratively discussed "schema". With a more powerful sensemaking tool, users would be able to collect larger chunks of information (*e.g.*, paragraphs around a selected word, map views, table columns, or pre-selected visualizations) into a shoebox. Afterwards, they could manually filter this information into an evidence file, which would be presented to the partner. Together, they could now merge their evidence files to a combined schema and subsequently to a presentation.

As extension of our series of experiments, it would be interesting to perform longitudinal observations of window management techniques for complex information management activities. As most of our techniques are implemented as plugins for the popular Compiz window manager, we can reach a large user base for long-term logging-based studies or informal feedback. We therefore aim to make our extensions publicly available to the community.

Part V

Summary

Chapter 10

Discussion and Future Directions

The research hypothesis of this thesis is formulated as follows: *Incorporating knowledge of the physical environment and window content in WIMP interfaces will support users in discovering, accessing, manipulating, and sharing information.* Various research prototypes have been designed and developed, and evaluations have been conducted to test this hypothesis. In the following, these research prototypes will be discussed with respect to the hypothesis and suggestions for future research directions will be presented.

10.1 Information Discovery

In this thesis, two window management techniques for information discovery have been presented: visual links and window uncovering.

Visual links across applications (Section 8.1) are a cue-based focus and context technique supporting the user to distinguish important from unimportant information in a display environment cluttered with information. The technique derives knowledge from two layers: the **window** layer to determine which regions of a window are relevant with respect to the user's current selection and the **environment** layer by distinguishing input by different users and spatially aware links routing across discontinuous displays. Visual links are then rendered as line connections on top of the screen content and important content in the windows is visually highlighted.

The purpose of visual links is to support the user in filtering a large amount of visible information in the environment, as well as to discover invisible information. According to the taxonomy of invisible information presented in Chapter 1, visual links thereby cover multiple aspects (*cf.*, Table 1.1): They visualize diminished information on the window level (*e.g.*, a location on a map despite a coarse zoom level) and on the environment level (the existence of information at a far distance, where the user cannot identify the information any more). In a large display environment, they furthermore indicate the existence of relevant information outside the user's field of view and hidden behind physical objects. In combination with the arrow-based off-screen visualization, they also help to

	Window	Screen	Environment
Occluded	–	uncovering windows	visual links
Cropped	off-screen arrow	–	visual links
Diminished	visual links	–	visual links

Table 10.1: Overview how our proposed window management techniques help to reveal invisible information, according to our taxonomy in Table 1.1. Mind that the primary purpose of visual links is to filter *visible* information.

discover information cropped at the window level, *e.g.*, text in a long document which requires the user to scroll, or a map location outside the current map view.

Results from two exploratory evaluations (Section 9.1 and Section 9.2) indicate that visual links supported our users in filtering a large amount of visible information on the screen. In a collaborative setting, visual links also reduced the necessity to modify shared window content, as diminished information on the window level – caused by a coarse zoom level – could be discovered without changing the visualization view.

In their current implementation, visual links do not support discovery of content occluded on the window or screen level (*i.e.*, information hidden behind tabs or top level windows). However, using importance-driven compositing window management in combination with links to occluded content would resolve this issue: Linked items are assigned a very high importance in the window importance map and are therefore most likely to be revealed using importance-driven compositing. Also, the current visual links routing mechanism does not consider minimized windows, even though minimized applications report user selections to the management application and the window state can be inferred from the window manager. Thus, adding an additional highlighting technique for guiding the user's attention to invisible information on the screen level, would be a desirable future extension for visual linking, as illustrated in Figure 10.1.

In contrast to visual links, **window uncovering** (Section 8.2) derives knowledge from

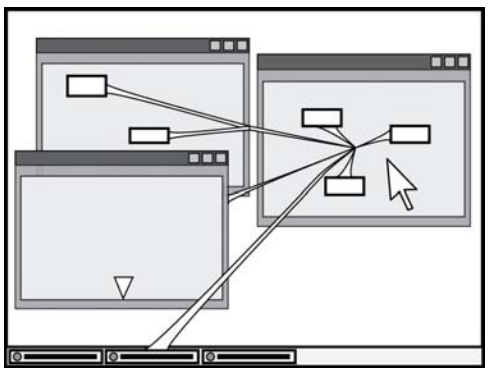


Figure 10.1: Conceptual sketch of extended visual links on a single display: in addition to the connection lines between highlighted window regions and windows with cropped elements (indicated by off-screen arrows), visual links could also infer knowledge of the window states and indicate the existence of highlighted elements in minimized windows.

the **window** and the **screen** level: On the window level, important window regions are determined based on the analysis of image-based window importance maps, while window occlusions are detected by the window manager on the screen level. Window uncovering then optimizes the spatial window layout on the screen level and applies pixel-wise transparencies to the individual windows to reveal occluded content.

Uncovering windows supports information discovery by temporarily increasing the amount of visible information on the screen, which the user has to filter mentally, *i.e.*, by sequentially scanning the entire display. It thereby supports the user in discovering occluded screen content (*cf.*, Table 10.1).

Results from a comparative experiment (Section 9.3) show that users were indeed faster to find occluded window content, compared to sequential Alt+tab switching or *free-space transparency* [123], which also applies content-aware transparencies to occluder windows but does not optimize the spatial window layout on the screen level. However, this advantage has only been observed for discovering fine-grained content, such as text, that was not readable in the scaled window thumbnails of the Alt+tab menu. To detect clearly visible and easily recognizable content, such as images, uncovering windows did not significantly improve user performance.

Window uncovering was evaluated on a small monitor, where screen size restrictions limit the amount of simultaneously visible information and thereby lead to an increased amount of occluded content. However, in large display environments, the more challenging aspect is filtering a large amount of visible information [234]. The above mentioned combination with visual links could help to guide the user's attention to relevant uncovered items.

In the future, the concept of visual links should not only be used for synchronized highlighting of user-selected data, but could also be employed to enhance window switching

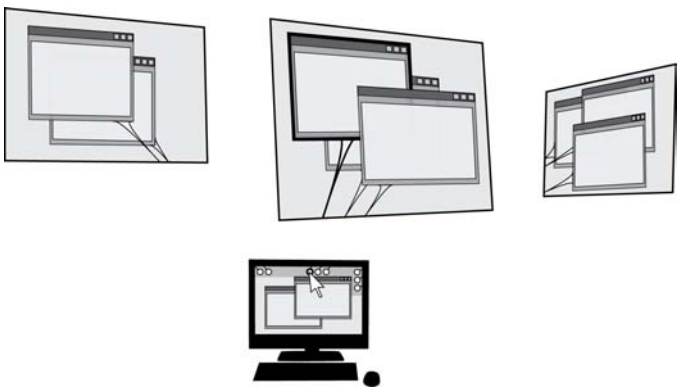


Figure 10.2: Conceptual sketch of visual links to windows on remote displays: on the user's home display, window icons are aligned along the display boundary. Visual links connect these icons to the respective remote windows. Pulling an icon towards the home display center relocates the associated remote window to the home display. Clicking on an icon teleports the pointer to the remote window.

techniques on large or discontinuous displays. As illustrated in Figure 10.2, cross-display links could help the user to discover remote windows – similar to the trail-based focus window visualizations by Hoffmann *et al.* [108]. However, these links to remote windows should not only serve as visualization, but should also be coupled to an interaction technique to quickly access content in remote windows.

10.2 Information Access

As stated in the previous section, **window uncovering** facilitates knowledge from the **window** and **screen** level to support discovery of occluded information. Window uncovering also enables information access, as it not only visually reveals occluded content, but also allows the user to interact with this revealed content.

In an experiment (Section 9.3), users were faster conducting simple interaction tasks in initially occluded windows, compared to conventional Alt+tab window switching, as well as *free-space transparency* [123], which only visually revealed occluded content but required explicit window switching to let users access the content. This ability for combined information discovery and access was one of the most appreciated aspects of window uncovering, according to informal user feedback.

However, information access requires the user to move the pointer to the respective window – like in most other window occlusion management techniques. In large display environments, access to distant information in remote windows is more physically and mentally demanding – an aspect that has not been addressed by the window uncovering technique.

We therefore developed and evaluated different **cross-display navigation techniques** (Chapter 5) to access remote items in a large, discontinuous display environment. All but one of these techniques facilitate knowledge from the **environment** layer to ensure spatially consistent navigation across display boundaries. Hypothetically, this spatial awareness should cognitively support the user when accessing information on a remote display using the mouse pointer.

According to our experiments (Chapter 6), spatial awareness seems to be only required in very compact and straight-forward settings. In more complex settings, users do not perceive the environment as spatially continuous any more. In this case, they subjectively prefer more "discrete" interaction techniques to forward input control across displays.

Currently, our navigation techniques only consider information from the environment layer. However, information from the screen or window layer could be similarly incorporated for mouse pointer navigation. Consider the conceptual sketch in Figure 6.13: combining spatial knowledge of the environment (*i.e.*, the visual navigation space) and knowledge of window locations on the screen may be used to determine optimal outcome positions when warping the pointer across display-less space. Window content awareness could furthermore enhance on-display pointing performance when considering important window regions in the device space. Based on window importance maps or synchronized

highlight regions, selective window regions can be made "sticky" by dynamically adjusting the C/D gain, such as proposed for *Semantic Pointing* [42] or *Sticky Widgets* [151].

In the future, it will be important to also consider cross-display window relocation to improve information access. In particular, *remote getting* [166], *i.e.*, relocating distant content to physically closer display areas for more convenient information access, has to be considered in this context. According to informal user feedback in an exploratory evaluation (Section 9.4), being able to quickly move windows back from a "context area" to a closer display location was assessed as one of the most important future window management techniques for large, irregular displays. As mentioned before, techniques to access remote content should be coupled with information discovery support. For instance, the concept to use visual links for discovering windows on remote displays, illustrated in Figure 10.2, could be extended to allow the user to quickly relocate a remote window to a convenient display space by simply "pulling" the cross-display link to the local display. Conversely, the user may wish to keep the window layout unchanged (for instance to keep interference with collaborators to a minimum) and rather wants to "teleport" the pointer to the remote window by clicking on the local window icon. Similarly, a world-in-miniature technique could allow the user to zoom into a remote display to discover occluded windows there and to subsequently relocate the discovered remote window to the local display with a simple button press or short mouse gesture – similarly as implemented in *SEAPort* [35] or the world-in-miniature for Compiz by Bayer [24].

10.3 Information Manipulation

Display-adaptive window management (Section 8.3) incorporates knowledge from all three layers to support information manipulation on large, irregular displays: Information about the physical display form factors is derived from the **environment** layer, while the **screen** layer provides window occlusion information, and the **window** layer informs about the importance of the window content. Display-adaptive window management facilitates this information to optimize the spatial window layout to increase the amount of important information on the screen and to minimize the amount of content located at inconvenient display locations. In contrast to window uncovering, it does not conduct any window-level operations (*i.e.*, see-through compositing) to increase content visibility.

Results from an initial exploratory evaluation (Section 9.4) showed that most users appreciated the ability to automatically resolve occlusions on demand. However, automatically snapping the windows to usable display areas was perceived as either unnecessary or even disturbing, when users explicitly facilitated the irregular display boundaries to arrange irrelevant information.

Informal user feedback and observations of this experiment lead to the conclusion that display-adaptive window management needs to be much more powerful than originally anticipated. As users explicitly facilitated the physical discontinuities of the display to sort and arrange their information, incorporating knowledge of the display environment

to automate certain window manager operations seems to be indeed a suitable approach. However, users demanded extended support for *remote putting* [166], *i.e.*, for easily placing windows in remote areas. They suggested two different approaches for efficient window relocation: The more accepted suggestion was to *throw* [88] windows to remote display areas with a simple mouse gesture. Spatial awareness of the environment is necessary to determine the suitable target display region and display-geometry snapping would ensure that the window is placed within a suitable display location (illustrated in Figure 10.3(a)). An alternative suggestion was to embed a relocation button into the window title bar. Again, awareness of the spatial display and user arrangement could help to categorize suitable display locations. Users could then choose their anticipated relocation purpose, such as *handing off* the window to a collaborator, *depositing* the window at a context region, or *publishing* it for joint discussion (inspired by basic *mechanics of collaboration* [179]). The system would then analyze the environment (*e.g.*, visibility for the users, size, and resolution of the potential target display) and the screen layer (*e.g.*, whether someone is currently interacting and if important windows would be occluded) to detect the most suitable target display candidate. Figure 10.3(b) illustrates potential relocation targets for depositing and publishing. However, users commented that pressing a button would have caused more effort than dragging the window – at least with respect to the rather small device space in our experiment.

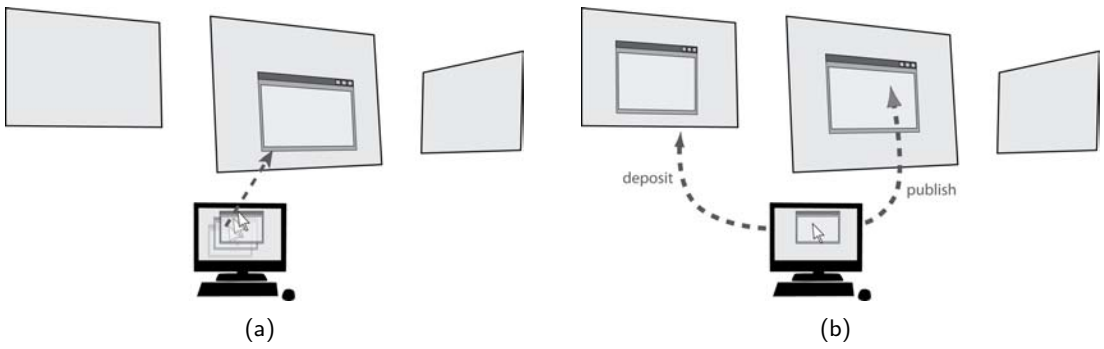


Figure 10.3: Conceptual sketches for window relocation in emerging display environments: (a) "throwing" windows to a remote display and (b) explicitly triggered relocation techniques.

Display-adaptive window management is only a first step towards more suitable window management for emerging display environments. It supports information manipulation on irregularly projected, vertical displays and could serve as foundation for window management in discontinuous MDEs, such as sketched in Figure 10.3. However, other emerging display factors, such as horizontal displays or small mobile devices, have not been addressed. Apart from technical challenges (as shortly discussed at the end of Chapter 8), window management for tabletop displays probably needs to be significantly different from the window management concepts we know today. Interaction problems, such as targeting difficulties with the mouse pointer observed in previous research (*e.g.*, [83]) and in one of

our experiments (Section 6.1), and challenges for information presentation, such as readability aspects [263], indicate that conventional WIMP interfaces cannot easily be adapted for tabletop displays. We therefore need to find alternative ways to window management, but also window content manipulation, on tabletop displays. Simple touch-sensitive surfaces where captured input events are simply mapped to conventional pointer movements and clicks (*e.g.*, as the unified software architecture for tabletop displays based on Compiz by Cheng *et al.* [58]), are already an improvement to mouse-based interaction, according to our initial experiences. However, for tabletop displays we probably need to move further away from conventional WIMP interfaces to truly meet the users' needs. A promising direction are tangible user interfaces for window management, as proposed by Holman *et al.* [109], where the technical feasibility for integration in the Compiz window manager has already been demonstrated [267].

The next logical step would be not only to adjust the window's placement, size, rotation, or opacity on the screen, but to actually adjust the window's content to the new display and viewing arrangements. In related research, examples for context-sensitive information visualization have been demonstrated in smart environments [84]. This topic goes beyond the WIMP interface layer and was therefore not handled in the scope of this thesis.

10.4 Information Sharing

Collaborative information linking (Section 8.1.2) facilitates personalized sets of visual links across applications to synchronize individual user selections across application boundaries. Like conventional visual links, it derives knowledge from the **window** layer (synchronized highlight regions) and the **environment** layer (distinguishing multi-user input and analyzing the spatial display arrangement for cross-display links routing).

In a collaborative setting, the purpose of links is not solely to support information discovery. It also helps users to maintain awareness of other users' activities by visualizing their current selections on shared application windows. It furthermore supports users in comparing their individual selections on a shared view, which may trigger further discussions.

In an exploratory evaluation (Section 9.2), groups conducting information analysis in a mixed-focus collaboration style [94] facilitated information linking for individual information discovery, and subsequently used links to replay their investigation steps to the partner for joint discussion. The foundation for this replay activity was enabled by a simple selection bookmarking tool or a manually constructed text document summarizing the individually collected findings in some sort of "shoebox" (*cf.*, Figure 1.5). User feedback suggests that tools for storing selections and findings are indeed useful but need to be much more powerful.

In contrast to collaborative information linking, **polarization-based interfaces** in combination with optical magic lenses (Section 8.4) provide personalized views on a phys-

ically shared display without altering the shared view. In contrast to all other interaction and presentation techniques presented in this thesis, polarization-based interfaces facilitate only passive knowledge from the **environment** layer by exploiting purely optical properties of polarized light. As a result, information filtering techniques normally requiring constant tracking of users and handheld lens devices can be provided without any software modifications of legacy applications. With this light-weight infrastructure, interaction with shared display content does not cause interference with the collaborators. Conceptually, we also proposed the incorporation of the **screen** layer to extend our concept to *single-display privacyware* [219], differentiating between shared and private windows, and to provide personalized highlighting, like visual links.

Co-located collaboration has not been investigated deeply in this thesis, although it is a very wide research area with countless facets. Co-located collaboration has been kept in mind for our presented software infrastructure and (most of) our research prototypes, but we only touched upon collaboration by facilitating personalized visual links across applications and polarization-based interfaces for collaborative information analysis tasks. Our findings in terms of co-located collaboration therefore should be seen as purely exploratory.

In the future, window relocation facilities exploiting spatial knowledge of the environment, as presented in the previous section (*cf.*, Figure 10.3), may furthermore support information sharing activities using conventional WIMP interfaces by enabling spontaneous discussions and flexible information exchange. Similarly, being able to quickly collect diverse window content into a consistent form to share the findings with the partner at a later stage was seen as useful – even though users expect much more powerful tools. A particularly important aspect is that window manager operations on shared display spaces should be very subtle to decrease interference.

Finally, information sharing should go beyond WIMP interfaces and should be explicitly supported by the applications themselves. As a next step, applications could adapt their representation and displayed data items to the active user. Streit *et al.* [229] addressed this issue theoretically for information visualization, where *multi-level interaction* allows to seamlessly switch between representations of different application levels, which is triggered by a change of owner. A system infrastructure using polarization-based interfaces is particularly useful to support such personalized views, where a switch between the views could be simply triggered by facilitating the tangible magic lens.

Chapter 11

Conclusion

The research prototypes for window-based information management in this thesis have shown that boundaries of conventional WIMP interfaces can be successfully pushed towards many varieties of emerging display environments. It has been demonstrated that future WIMP interfaces can incorporate more knowledge than just the screen's virtual outline, described as simple rectangle, the location of a single pointer, and the location and size of rectangular, opaque windows and other desktop elements, like menus or icons. We summarized this type of knowledge as *screen layer*, as it only contains basic knowledge of simple screen elements, without further information about the display's properties in the environment or the screen elements' content.

In this thesis, we have extended WIMP interfaces to incorporate knowledge from two additional layers:
environment: information about the display form factors, the spatial arrangement of displays in the environment, and users operating them, and
windows: information about the individual windows' content.

To retrieve this information, we designed and developed adequate system infrastructure to build low-cost spatially aware (multi-)display environments (Part II). In addition, infrastructure for multi-user mouse pointer interaction incorporating this spatial knowledge (Chapter 5) and window management incorporating spatial knowledge, but also information about the window content, has been presented (Chapter 7). Based on this infrastructure, mouse pointer navigation techniques have been developed (Chapter 5) and novel window management techniques have been designed and implemented (Chapter 8). These techniques have been evaluated to address different aspects of the research hypothesis of this thesis:

Incorporating knowledge of the physical environment and window content in WIMP interfaces will support users in discovering, accessing, manipulating, and sharing information.

We could show that incorporating knowledge of the **environment** – in particular of the spatial display arrangement – can have a positive impact on the performance of

pointing tasks for remote information **access** in discontinuous MDEs. However, it seems that with a certain degree of complexity concerning the display arrangement, the users did not perceive the navigation space as spatially continuous any more. In these cases, spatial awareness of the navigation technique did not provide any advantage. In contrast, compensating for an increased visual-device space mismatch (with the cost of a larger travel distance) by incorporating spatial knowledge decreased the performance.

We could furthermore demonstrate that facilitating knowledge of the **window** content in the **window manager** can support users to **discover** information, **access** occluded window content, and to **share** findings among collaborators. While visual links across applications are suitable to visually filter a large amount of visible information on large displays, window uncovering supports the user in discovering occluded content on conventionally sized monitors. Users particularly appreciated window uncovering as a unified interface to reveal occluded information, which also allows for direct access to the otherwise occluded information. For groups working in a mixed-focus collaboration style [94], we could furthermore observe that visual links in combination with storage mechanisms support users in sharing (or "replaying") individual findings with their partner in a co-located collaborative setting.

Finally, incorporating knowledge of the **environment** – more specifically, the physical display form factor – in the **window manager** seems to be a promising direction to facilitate information **manipulation** on irregularly shaped projected displays. Users invested a significant amount of time to establish spatial window layouts supporting direct visual comparisons, for sorting, and for categorizing individual windows – activities that have been identified as some sort of *schematizing* in the sensemaking process [4, 182]. Users thereby explicitly facilitated the physical irregularities of the display to create meaningful layouts. Display-adaptivity features helped users to avoid window placement across physical discontinuities and to resolve occlusions. However, users indicated that display-adaptivity needs to be much more powerful to fully exploit the display form factors for their sensemaking activities.

For future designers and developers of WIMP interfaces, our findings imply that it is indeed feasible to extend conventional WIMP interfaces to support information management in emerging display environments. We could also demonstrate that maintaining spatial awareness of the environment is possible without special sensory equipment or high-end display hardware in many display configurations, such as illustrated in Figure 11.1. Dynamic spatial awareness facilitating tracking hardware will be necessary if mobile devices are employed, or if the users frequently change location. We also discussed different window manager extensions to access information about the window content in WIMP interfaces. Our presented window management techniques illustrate that powerful interaction and information presentation techniques can build upon non-invasive and minimally invasive content-awareness approaches.

In the future, information about the environment layer may not only be facilitated by WIMP interfaces, but also by custom applications. Our Deskotheque environment provides

(a) (b)

Figure 11.1: Examples for multi-display environments which are suitable for a static environmental model: personal-device centric operation with (a) a single shared display and (b) displays of different form factors arranged in front of the users.

an API to access information about the environment (*e.g.*, users triggering an input event and geometric properties of the displays) for custom application development. Future applications running in such an extended WIMP environment should consider adjusting their information representation dynamically to the interacting user(s) and the display factors. Using such an infrastructure, special-purpose software dynamically adapted to the environment can co-exist with unmodified applications that benefit from application-transparent and environment-aware pointer navigation and window management. Information about the environment should also be facilitated by GUI toolkits to support the creation of adaptive user interfaces, as proposed, for instance, by Myers *et al.* [161] or Gajos and Weld [86].

Furthermore, easy-to-access interfaces to synchronize user activities across window (and application) boundaries are necessary to make content-aware visualization and interaction techniques widely available. Having such an infrastructure and applications making use of it, content-awareness cannot only be facilitated for information filtering and occlusion management, such as presented in this thesis, but could also improve the support for (collaborative) sensemaking.

Bibliography

[1] Anand Agarawala and Ravin Balakrishnan. Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. In *Proc. CHI 2006*, pages 1283–1292. ACM, 2006.

[2] Christopher Ahlberg. Spotfire: an information exploration environment. *SIGMOD Rec.*, 25:25–29, December 1996.

[3] Saleema Amershi and Meredith Ringel Morris. Cosearch: a system for co-located collaborative web search. In *Proc. CHI 2008*, pages 1647–1656. ACM, 2008.

[4] Christopher Andrews, Alex Endert, and Chris North. Space to think: large high-resolution displays for sensemaking. In *Proc. CHI 2010*, pages 55–64. ACM, 2010.

[5] Caroline Appert, Olivier Chapuis, and Michel Beaudouin-Lafon. Evaluation of pointing performance on screen edges. In *Proc. AVI 2008*, pages 119–126. ACM, 2008.

[6] Mark Ashdown, Matthew Flagg, Rahul Sukthankar, and James M. Rehg. A Flexible Projector-Camera System for Multi-Planar Displays. In *Proc. CVPR 2004*, volume 2, pages 165–172. IEEE Computer Society, 2004.

[7] Mark Ashdown, Kenji Oka, and Yoichi Sato. Combining head tracking and mouse input for a gui on multiple monitors. In *Proc. CHI 2005*, pages 1188–1191. ACM, 2005.

[8] Mark Ashdown and Peter Robinson. Escritoire: A personal projected display. *IEEE MultiMedia*, 12(1):34–42, 2005.

[9] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. *ACM Trans. Graph.*, 26, July 2007.

[10] Greg J. Badros, Jeffrey Nichols, and Alan Borning. Scwm: An extensible constraint-enabled window manager. In *Proc. USENIX 2001*, pages 225–234. USENIX Association, 2001.

[11] Ronald M. Baecker. *Readings in human-computer interaction: toward the year 2000.* Morgan Kaufmann, 1995.

[12] Ravin Balakrishnan. "beating" fitts' law: virtual enhancements for pointing facilitation. *Int. J. Hum.-Comput. Stud.*, 61(6):857–874, 2004.

[13] Robert Ball and Chris North. An analysis of user behavior on high-resolution tiled displays. In *In Proc. INTERACT 2005*, pages 350–363. Springer, 2005.

[14] Robert Ball and Chris North. Effects of tiled high-resolution display on basic visualization and navigation tasks. In *Ext. Abstracts CHI 2005*, pages 1196–1199. ACM, 2005.

[15] Liam Bannon, Allen Cypher, Steven Greenspan, and Melissa L. Monty. Evaluation and analysis of users' activity organization. In *Proc. CHI 1983*, pages 54–57. ACM, 1983.

[16] Aaron Barsky, Tamara Munzner, Jennifer Gardy, and Robert Kincaid. Cerebral: Visualizing multiple experimental conditions on a graph with biological context. *IEEE TVCG*, 14:1253–1260, November 2008.

[17] Patrick Baudisch, Edward Cutrell, Mary Czerwinski, Daniel C. Robbins, Peter Tandler, Benjamin B. Bederson, and A. Zierlinger. Drag-and-pop and drag-and-pick: Techniques for accessing remote screen content on touch- and pen-operated systems. In *Proc. INTERACT 2003*, pages 57–64, 2003.

[18] Patrick Baudisch, Edward Cutrell, Ken Hinckley, and Robert Gruen. Mouse ether: accelerating the acquisition of targets across multi-monitor displays. In *Ext. Abstracts CHI 2004*, pages 1379–1382. ACM, 2004.

[19] Patrick Baudisch, Edward Cutrell, and George G. Robertson. High-density cursor: a visualization technique that helps users keep track of fast-moving mouse cursors. In *Proc. INTERACT 2003*, pages 236–243, 2003.

[20] Patrick Baudisch, Nathaniel Good, Victoria Bellotti, and Pamela Schraedley. Keeping things in context: a comparative evaluation of focus plus context screens, overviews, and zooming. In *Proc. CHI 2002*, pages 259–266. ACM, 2002.

[21] Patrick Baudisch, Nathaniel Good, and Paul Stewart. Focus plus context screens: combining display technology with visualization techniques. In *Proc. UIST 2001*, pages 31–40. ACM, 2001.

[22] Patrick Baudisch and Carl Gutwin. Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. In *Proc. CHI 2004*, pages 367–374. ACM, 2004.

[23] Patrick Baudisch and Ruth Rosenholtz. Halo: a technique for visualizing off-screen objects. In *Proc. CHI 2003*, pages 481–488. ACM, 2003.

[24] Joris Jan Bayer. A world-in-miniature for multi-display environments: Implementation of a compiz-based window relocator. Bachelor's thesis, Institute for Computer Graphics and Vision, Graz University of Technology, 2011.

[25] Michel Beaudouin-Lafon. Novel interaction techniques for overlapping windows. In *Proc. UIST 2001*, pages 153–154. ACM, 2001.

[26] Benjamin B. Bederson and James D. Hollan. Pad++: a zooming graphical interface for exploring alternate interface physics. In *Proc. UIST 1994*, pages 17–26. ACM, 1994.

[27] Blaine A. Bell and Steven K. Feiner. Dynamic space management for user interfaces. In *Proc. UIST 2000*, pages 239–248. ACM, 2000.

[28] Hrvoje Benko and Steven Feiner. Multi-monitor mouse. In *Ext. Abstracts CHI 2005*, pages 1208–1211. ACM, 2005.

[29] Hrvoje Benko and Steven Feiner. Pointer warping in heterogeneous multi-monitor environments. In *Proc. GI 2007*, pages 111–117. ACM, 2007.

[30] Anastasia Bezerianos and Ravin Balakrishnan. The vacuum: facilitating the manipulation of distant objects. In *Proc. CHI 2005*, pages 361–370. ACM, 2005.

[31] Anastasia Bezerianos, Pierre Dragicevic, and Ravin Balakrishnan. Mnemonic rendering: an image-based approach for exposing hidden changes in dynamic displays. In *Proc. UIST 2006*, pages 159–168. ACM, 2006.

[32] Xiaojun Bi and Ravin Balakrishnan. Comparing usage of a large high-resolution display to single or dual desktop displays for daily work. In *Proc. CHI 2009*, pages 1005–1014. ACM, 2009.

[33] Jacob T. Biehl and Brian P. Bailey. ARIS: An Interface for Application Relocation in an Interactive Space. In *Proc. GI 2004*, pages 107–116. Canadian Human-Computer Communications Society, 2004.

[34] Jacob T. Biehl and Brian P. Bailey. Comparing an iconic interface to a text-based and virtual paths interface for effective interaction in an interactive workspace. In *Proc. ED-MEDIA 2006*, pages 581–586, 2006.

[35] Jacob T. Biehl and Brian P. Bailey. Improving scalability and awareness in iconic interfaces for multiple-device environments. In *Proc. AVI 2006*, pages 91–94. ACM, 2006.

[36] Jacob T. Biehl and Brian P. Bailey. Interfaces for managing applications and input in multi-device environments. In *CHI Workshop on Information Visualization and Interaction Techniques for Collaboration across Multiple Displays*, 2006.

[37] Jacob T. Biehl, William T. Baker, Brian P. Bailey, Desney S. Tan, Kori M. Inkpen, and Mary Czerwinski. IMPROMPTU: A New Interaction Framework for Supporting Collaboration in Multiple Display Environments and Its Field Evaluation for Co-located Software Development. In *Proc. CHI 2008*, pages 939–948. ACM, 2008.

[38] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: the see-through interface. In *Proc. SIGGRAPH 1993*, pages 73–80. ACM, 1993.

[39] Oliver Bimber, Andreas Emmerling, and Thomas Klemmer. Embedded entertainment with smart projectors. *Computer*, 38:48–55, January 2005.

[40] Oliver Bimber, Gordon Wetzstein, Andreas Emmerling, and Christian Nitschke. Enabling view-dependent stereoscopic projection in real environments. In *Proc. ISMAR 2005*, pages 14–23. IEEE Computer Society, 2005.

[41] Jeremy P. Birnholtz, Tovi Grossman, Clarissa Mak, and Ravin Balakrishnan. An exploratory study of input configuration and group process in a negotiation task using a large display. In *Proc. CHI 2007*, pages 91–100. ACM, 2007.

[42] Renaud Blanch, Yves Guiard, and Michel Beaudouin-Lafon. Semantic pointing: improving target acquisition with control-display ratio adaptation. In *Proc. CHI 2004*, pages 519–526. ACM, 2004.

[43] Sara A. Bly and Jarrett K. Rosenberg. A comparison of tiled and overlapping windows. *SIGCHI Bull.*, 17(4):101–106, 1986.

[44] Kellogg S. Booth, Brian D. Fisher, Chi Jui Raymond Lin, and Ritchie Argue. The "mighty mouse" multi-screen collaboration tool. In *Proc. UIST 2002*, pages 209–212. ACM, 2002.

[45] Sebastian Boring, Dominikus Baur, Andreas Butz, Sean Gustafson, and Patrick Baudisch. Touch projector: mobile interaction through video. In *Proc. CHI 2010*, pages 2287–2296. ACM, 2010.

[46] Jean-Yves Bouguet. Camera calibration toolbox for matlab. URL: http://www.vision.caltech.edu/bouguetj/calib_doc/, Last access: December 2010.

[47] Gary Bradski and Adrian Kaehler. *Learning OpenCV: computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

[48] Michael S. Brown, Aditi Majumder, and Ruigang Yang. Camera-Based Calibration Techniques for Seamless Multiprojector Displays. *IEEE TVCG*, 11(2):193–206, 2005.

[49] Michael S. Brown and W. Brent Seales. A practical and flexible tiled display system. In *Proc. Pacific Graphics 2002*, pages 194–203, 2002.

[50] Kevin F. Bury and Michael J. Darnell. Window management in interactive computer systems. *SIGCHI Bull.*, 18:65–66, October 1986.

[51] S. K. Card, M. Pavel, and J. E. Farrell. Human-computer interaction. chapter Window-based computer dialogues, pages 456–460. Morgan Kaufmann Publishers Inc., 1987.

[52] Stuart K. Card and Austin Henderson, Jr. A multiple, virtual-workspace interface to support user task switching. In *Proc. CHI 1987*, pages 53–59. ACM, 1987.

[53] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[54] M. Sheelagh T. Carpendale and Catherine Montagnese. A framework for unifying presentation space. In *Proc. UIST 2001*, pages 61–70. ACM, 2001.

[55] Olivier Chapuis and Nicolas Roussel. Metisse is not a 3D Desktop! In *Proc. UIST 2005*, pages 13–22. ACM, 2005.

[56] Olivier Chapuis and Nicolas Roussel. Copy-and-paste between overlapping windows. In *Proc. CHI 2007*, pages 201–210. ACM, 2007.

[57] Han Chen, Rahul Sukthankar, Grant Wallace, and Tat-Jen Cham. Calibrating scalable multi-projector displays using camera homography trees. In *Proc. CVPR 2001*, pages 9–14. IEEE Computer Society, 2001.

[58] Kelvin Cheng, Benjamin Itzstein, Paul Sztajer, and Markus Rittenbruch. A unified multi-touch & multi-pointer software architecture for supporting collocated work on the desktop. Technical Report ATP-2247, NICTA, Sydney, Australia, 2010.

[59] Andy Cockburn, Amy Karlson, and Benjamin B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.*, 41:2:1–2:31, January 2009.

[60] Ellis S. Cohen, Edward T. Smith, and Lee A. Iverson. Constraint-based tile windows. *IEEE Comput. Graph. Appl.*, 6(5):35–45, 1986.

[61] Christopher Collins and Sheelagh Carpendale. Vislink: Revealing relationships amongst visualizations. *IEEE TVCG*, 13:1192–1199, November 2007.

[62] Gregorio Convertino, Jian Chen, Beth Yost, Young-Sam Ryu, and Chris North. Exploring context switching and cognition in dual-view coordinated visualizations. In *Proc. CMV 2003*, pages 55–60. IEEE Computer Society, 2003.

[63] Gregorio Convertino, Craig H. Ganoe, Wendy A. Schafer, Beth Yost, and John M. Carroll. A multiple view approach to support common ground in distributed and synchronous geo-collaboration. In *Proc. CMV 2005*, pages 121–132. IEEE Computer Society, 2005.

[64] Daniel Cotting, Henry Fuchs, Remo Ziegler, and Markus H. Gross. Adaptive Instant Displays: Continuously Calibrated Projections Using Per-Pixel Light Control. *Computer Graphics Forum*, 24(3):705–714, 2005.

[65] Daniel Cotting and Markus Gross. Interactive Environment-Aware Display Bubbles. In *Proc. UIST 2006*, pages 245–254. ACM, 2006.

[66] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *Proc. SIGGRAPH 1993*, pages 135–142. ACM Press, 1993.

[67] Mary Czerwinski, Eric Horvitz, and Susan Wilhite. A diary study of task switching and interruptions. In *Proc. CHI 2004*, pages 175–182. ACM, 2004.

[68] Mary Czerwinski, George Robertson, Brian Meyers, Greg Smith, Daniel Robbins, and Desney Tan. Large Display Research Overview. In *Ext. Abstracts CHI 2006*, pages 69–74. ACM, 2006.

[69] Mary Czerwinski, Greg Smith, Tim Regan, and Brian Meyers. Toward characterizing the productivity benefits of very large displays. In *Proc. INTERACT 2003*, pages 9–16. IOS Press, 2003.

[70] Jr. D. Austin Henderson 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.

[71] Mark Dokter. Synergy+mpx: Towards multi-user interaction in multi-display environments. Bachelor thesis, Institute for Computer Graphics and Vision, Graz University of Technology, Austria, 2010.

[72] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware: some issues and experiences. *Commun. ACM*, 34(1):39–58, 1991.

[73] Niklas Elmqvist and Philippas Tsigas. A taxonomy of 3d occlusion management for visualization. *IEEE TVCG*, 14:1095–1109, September 2008.

[74] Katherine Everitt, Chia Shen, Kathy Ryall, and Clifton Forlines. Multispace: Enabling electronic document micro-mobility in table-centric, multi-device environments. In *Proc. TABLETOP 2006*, pages 27–34. IEEE Computer Society, 2006.

[75] Guillaume Faure, Olivier Chapuis, and Nicolas Roussel. Power tools for copying and moving: useful stuff for your desktop. In *Proc. CHI 2009*, pages 1675–1678. ACM, 2009.

[76] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, June 1981.

[77] Ken Fishkin and Maureen C. Stone. Enhanced dynamic queries via movable filters. In *Proc. CHI 1995*, pages 415–420. ACM Press/Addison-Wesley Publishing Co., 1995.

[78] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *J Exp Psychol*, 47(6):381–391, June 1954.

[79] Clifton Forlines, Alan Esenther, Chia Shen, Daniel Wigdor, and Kathy Ryall. Multi-user, multi-display interaction with a single-user, single-display geospatial application. In *Proc. UIST 2006*, pages 273–276. ACM, 2006.

[80] Clifton Forlines and Ryan Lilien. Adapting a single-user, single-display molecular visualization application for use in a multi-user, multi-display environment. In *Proc. AVI 2008*, pages 367–371. ACM, 2008.

[81] Clifton Forlines and Chia Shen. Dtlens: multi-user tabletop spatial data exploration. In *Proc. UIST 2005*, pages 119–122. ACM, 2005.

[82] Clifton Forlines, Chia Shen, Daniel Wigdor, and Ravin Balakrishnan. Exploring the effects of group size and display configuration on visual search. In *Proc. CSCW 2006*, pages 11–20. ACM, 2006.

[83] Clifton Forlines, Daniel Wigdor, Chia Shen, and Ravin Balakrishnan. Direct-touch vs. mouse input for tabletop displays. In *Proc. CHI 2007*, pages 647–656. ACM, 2007.

[84] Georg Fuchs, Conrad Thiede, Mike Sips, and Heidrun Schumann. Device-based adaptation of visualizations in smart environments. In *Proc. CoVIS Workshop, IEEE VisWeek 2009*, 2009.

[85] George W. Furnas. Generalized fisheye views. In *Proc. CHI 1986*, pages 16–23. ACM, 1986.

[86] Krzysztof Gajos and Daniel S. Weld. Supple: automatically generating user interfaces. In *Proc. IUI 2004*, pages 93–100. ACM, 2004.

[87] K. B. Gaylin. How are windows used? some notes on creating an empirically-based windowing benchmark task. In *Proc. CHI 1986*, pages 96–100. ACM, 1986.

[88] Jörg Geißler. Shuffle, throw or take it! working efficiently with an interactive wall. In *Proc. CHI 1998*, pages 265–266. ACM, 1998.

[89] Adele Goldberg and David Robson. *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., 1983.

[90] Saul Greenberg. Sharing views and interactions with single-user applications. In *Proc. COCS 1990*, pages 227–237. ACM, 1990.

[91] Jonathan Grudin. Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. In *Proc. CHI 2001*, pages 458–465. ACM, 2001.

[92] Sean Gustafson, Patrick Baudisch, Carl Gutwin, and Pourang Irani. Wedge: clutter-free visualization of off-screen locations. In *Proc. CHI 2008*, pages 787–796. ACM, 2008.

[93] Carl Gutwin, Jeff Dyck, and Chris Fedak. The effects of dynamic transparency on targeting performance. In *Proc. GI 2003*, pages 105–112, 2003.

[94] Carl Gutwin and Saul Greenberg. Design for individuals, design for groups: tradeoffs between power and workspace awareness. In *Proc. CSCW 1998*, pages 207–216. ACM, 1998.

[95] Carl Gutwin and Saul Greenberg. The mechanics of collaboration: Developing low cost usability evaluation methods for shared workspaces. In *Proc. WETICE 2000*, pages 98–103. IEEE Computer Society, 2000.

[96] Vicki Ha, Kori Inkpen, Jim Wallace, and Ryder Ziola. Swordfish: User Tailored Workspaces in Multi-Display Environments. In *Ext. Abstracts CHI 2006*, pages 1487–1492. ACM, 2006.

[97] Vicki Ha, Jim Wallace, Ryder Ziola, and Kori Inkpen. My mde: configuring virtual workspaces in multi-display environments. In *Ext. Abstracts CHI 2006*, pages 1481–1486. ACM, 2006.

[98] Beverly L. Harrison, Hiroshi Ishii, Kim J. Vicente, and William A. S. Buxton. Transparent layered user interfaces: an evaluation of a display design to enhance focused and divided attention. In *Proc. CHI 1995*, pages 317–324. ACM, 1995.

[99] Beverly L. Harrison, Gordon Kurtenbach, and Kim J. Vicente. An experimental evaluation of transparent user interface tools and information content. In *Proc. UIST 1995*, pages 81–90. ACM, 1995.

[100] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, February 1968.

[101] Kirstie Hawkey, Melanie Kellar, Derek Reilly, Tara Whalen, and Kori M. Inkpen. The proximity factor: impact of distance on co-located collaboration. In *Proc. GROUP 2005*, pages 31–40. ACM, 2005.

[102] Jeffrey Heer and Maneesh Agrawala. Design considerations for collaborative visual analytics. *Information Visualization*, 7(1):49–62, 2008.

[103] Jeffrey Heer, Stuart K. Card, and James A. Landay. prefuse: a toolkit for interactive information visualization. In *Proc. CHI 2005*, pages 421–430. ACM, 2005.

[104] Jeffrey Heer, Frank Ham, Sheelagh Carpendale, Chris Weaver, and Petra Isenberg. Information visualization. chapter Creation and Collaboration: Engaging New Audiences for Information Visualization, pages 92–133. Springer-Verlag, 2008.

[105] Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE TVCG*, 14:1189–1196, November 2008.

[106] Ken Hinckley. The human-computer interaction handbook. chapter Input technologies and techniques, pages 151–168. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 2003.

[107] Ken Hinckley, Gonzalo Ramos, Francois Guimbretiere, Patrick Baudisch, and Marc Smith. Stitching: pen gestures that span multiple displays. In *Proc. AVI 2004*, pages 23–31. ACM, 2004.

[108] Raphael Hoffmann, Patrick Baudisch, and Daniel S. Weld. Evaluating visual cues for window switching on large screens. In *Proc. CHI 2008*, pages 929–938. ACM, 2008.

[109] David Holman, Roel Vertegaal, Mark Altosaar, Nikolaus Troje, and Derek Johns. Paper windows: interaction techniques for digital paper. In *Proc. CHI 2005*, pages 591–599. ACM, 2005.

[110] Danny Holten and Jarke J. van Wijk. Force-directed edge bundling for graph visualization. In *Proc. EuroVis 2009*, pages 983 – 990, 2009.

[111] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *Proc. SIGGRAPH 2002*, pages 693–702. ACM, 2002.

[112] 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 *Proc. AVI 2004*, pages 32–39. ACM, 2004.

[113] Dugald Ralph Hutchings and John Stasko. Revisiting display space management: understanding current practice to inform next-generation design. In *Proc. GI 2004*, pages 127–134. Canadian Human-Computer Communications Society, 2004.

[114] Dugald Ralph Hutchings and John Stasko. Shrinking window operations for expanding display space. In *Proc. AVI 2004*, pages 350–353. ACM, 2004.

[115] Peter Hutterer and Bruce H. Thomas. Groupware Support in the Windowing System. In *Proc. AUIC 2007*, pages 39–46. Australian Computer Society, Inc., 2007.

[116] Keigo Iizuka. Cellophane as a half-wave plate and its use for converting a laptop computer screen into a three-dimensional display. *Review of Scientific Instruments*, 74(8):3636–3639, 2003.

[117] Kori Inkpen, Kirstie Hawkey, Melanie Kellar, Regan M, Karen Parker, Derek Reilly, Stacey Scott, and Tara Whalen. Exploring display factors that influence co-located collaboration: angle, size, number, and user arrangement. In *In Proc. HCI International 2005*, 2005.

[118] Pourang Irani, Carl Gutwin, and Xing Dong Yang. Improving selection of off-screen targets with hopping. In *Proc. CHI 2006*, pages 299–308. ACM, 2006.

[119] Petra Isenberg, Anastasia Bezerianos, Nathalie Henry, Sheelagh Carpendale, and Jean-Daniel Fekete. Coconuttrix: Collaborative retrofitting for information visualization. *Computer Graphics and Applications: Special Issue on Collaborative Visualization*, 29(5):44–57, September/October 2009.

[120] Petra Isenberg and Sheelagh Carpendale. Interactive tree comparison for co-located collaborative information visualization. *IEEE TVCG*, 13(6):1232–1239, November 2007.

[121] Petra Isenberg and Danyel Fisher. Collaborative brushing and linking for co-located visual analytics of document collections. In *Proc. EuroVis 2009*, volume 28, pages 1031–1038, June 2009.

[122] Petra Isenberg, Anthony Tang, and Sheelagh Carpendale. An exploratory study of visual information analysis. In *Proc. CHI 2008*, pages 1217–1226. ACM, 2008.

[123] Edward W. Ishak and Steven K. Feiner. Interacting with hidden content using content-aware free-space transparency. In *Proc. UIST 2004*, pages 189–192. ACM, 2004.

[124] Laurent Itti, Cristof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *PAMI*, 20(11):1254–1259, 1998.

[125] 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 *Proc. UIST 2003*, pages 159–168. ACM, 2003.

[126] Allan S. Jacobson, Andrew L. Berkin, and Martin N. Orton. Linkwinds: interactive scientific data analysis and visualization. *Commun. ACM*, 37:42–52, April 1994.

[127] Hao Jiang, Daniel Wigdor, Clifton Forlines, Michelle Borkin, Jens Kauffmann, and Chia Shen. Livolay: interactive ad-hoc registration and overlapping of applications for collaborative visual exploration. In *Proc. CHI 2008*, pages 1357–1360. ACM, 2008.

[128] Brad Johanson, Armando Fox, and Terry Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing*, 1(2):67–74, 2002.

[129] Brad Johanson, Greg Hutchins, Terry Winograd, and Maureen Stone. PointRight: Experience with Flexible Input Redirection in Interactive Workspaces. In *Proc. UIST 2002*, pages 227–234. ACM, 2002.

[130] Brad Johanson, Shankar Ponnekanti, Caesar Sengupta, and Armando Fox. Multibrowsing: Moving web content across multiple displays. In *Proc. UbiComp 2001*, pages 346–353. Springer-Verlag, 2001.

[131] Brian Johnson and Ben Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proc. VIS 1991*, pages 284–291. IEEE Computer Society Press, 1991.

[132] Jeff Johnson, Teresa L. Roberts, William Verplank, David C. Smith, Charles H. Irby, Marian Beard, and Kevin Mackey. The xerox star: A retrospective. *Computer*, 22:11–26, 28–29, September 1989.

[133] Tyler Johnson and Henry Fuchs. Real-time projector tracking on complex geometry using ordinary imagery. In *Proc. CVPR 2007*, page 1. IEEE Computer Society, 2007.

[134] Tyler Johnson and Henry Fuchs. A unified multi-surface, multi-resolution workspace with camera-based scanning and projector-based illumination. In *Proc. IPT-EGVE 2007*, 2007.

[135] Eser Kandogan and Ben Shneiderman. Elastic windows: evaluation of multi-window operations. In *Proc. CHI 1997*, pages 250–257. ACM, 1997.

[136] Daniel A. Keim. Information visualization and visual data mining. *IEEE TVCG*, 8:1–8, January 2002.

[137] Azam Khan, George Fitzmaurice, Don Almeida, Nicolas Burtnyk, and Gordon Kurtenbach. A remote control interface for large displays. In *Proc. UIST 2004*, pages 127–136. ACM, 2004.

[138] Azam Khan, Justin Matejka, George Fitzmaurice, and Gordon Kurtenbach. Spotlight: directing users' attention on large displays. In *Proc. CHI 2005*, pages 791–798. ACM, 2005.

[139] Robert Kosara, Silvia Miksch, and Helwig Hauser. Focus+Context taken literally. *IEEE Computer Graphics and Applications*, 22(1):22–29, 2002.

[140] J. Chris Lauwers and Keith A. Lantz. Collaboration awareness in support of collaboration transparency: requirements for the next generation of shared window systems. In *Proc. CHI 1990*, pages 303–311. ACM, 1990.

[141] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *Proc. SIGGRAPH 2002*, pages 362–371, 2002.

[142] Alexander Lex, Marc Streit, Ernst Kruijff, and Dieter Schmalstieg. Caleydo: Design and evaluation of a visual analysis framework for gene expression data in its biological context. In *Proc. PacificVis 2010*. IEEE Computer Society, 2010.

[143] Blair MacIntyre, Elizabeth D. Mynatt, Stephen Voida, Klaus M. Hansen, Joe Tullio, and Gregory M. Corso. Support For Multitasking and Background Awareness Using Interactive Peripheral Displays. In *Proc. UIST 2001*, pages 41–50, 2001.

[144] I. Scott MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *Hum.-Comput. Interact.*, 7(1):91–139, 1992.

[145] I. Scott MacKenzie, Tatu Kauppinen, and Miika Silfverberg. Accuracy measures for evaluating computer pointing devices. In *Proc. CHI 2001*, pages 9–16. ACM, 2001.

[146] Jock D. Mackinlay and Jeffrey Heer. Wideband displays: mitigating multiple monitor seams. In *Ext. Abstracts CHI 2004*, pages 1521–1524. ACM, 2004.

[147] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: detail and context smoothly integrated. In *Proc. CHI 1991*, pages 173–176. ACM, 1991.

[148] Jock D. Mackinlay, George G. Robertson, and Robert DeLine. Developing calendar visualizers for the information visualizer. In *Proc. UIST 1994*, pages 109–118. ACM, 1994.

[149] Thomas W. Malone. How do people organize their desks?: Implications for the design of office information systems. *ACM Trans. Inf. Syst.*, 1:99–112, January 1983.

[150] Richard Mander, Gitta Salomon, and Yin Yin Wong. A "pile" metaphor for supporting casual organization of information. In *Proc. CHI 1992*, pages 627–634. ACM, 1992.

[151] Regan L. Mandryk, Malcolm E. Rodgers, and Kori M. Inkpen. Sticky widgets: pseudo-haptic widget enhancements for multi-monitor displays. In *Ext. Abstracts CHI 2005*, pages 1621–1624. ACM, 2005.

[152] Regan L. Mandryk, Stacey D. Scott, and Kori M. Inkpen. Display factors influencing co-located collaboration. In *Proc. CSCW 2002*. ACM, 2002.

[153] Gloria Mark and Alfred Kobsa. The effects of collaboration and system transparency on cive usage: an empirical study and model. *Presence: Teleoper. Virtual Environ.*, 14(1):60–80, 2005.

[154] Allen R. Martin and Matthew O. Ward. High dimensional brushing for interactive exploration of multivariate data. In *Proc. VIS 1995*, page 271. IEEE Computer Society Press, 1995.

[155] Tara Matthews, Mary Czerwinski, George Robertson, and Desney Tan. Clipping lists and change borders: improving multitasking efficiency with peripheral information design. In *Proc. CHI 2006*, pages 989–998. ACM, 2006.

[156] Erick Mendez, Steven K. Feiner, and Dieter Schmalstieg. Focus and context in mixed reality by modulating first order salient features. In *Proc. SG 2010*, pages 232–243. Springer-Verlag, 2010.

[157] Tunu Miah and James L. Alty. Visual recognition of windows: effects of size variation and presentation styles. In *Proc. OzCHI 1998*, pages 72 –79. IEEE Computer Society, 1998.

[158] Meredith Ringel Morris, Andreas Paepcke, and Terry Winograd. Teamsearch: Comparing techniques for co-present collaborative search of digital media. In *Proc. TABLETOP 2006*, pages 97–104. IEEE Computer Society, 2006.

[159] Meredith Ringel Morris, Kathy Ryall, Chia Shen, Clifton Forlines, and Frederic Vernier. Beyond "social protocols": multi-user coordination policies for co-located groupware. In *Proc. CSCW 2004*, pages 262–265. ACM, 2004.

[160] Markus Murschitz. Deskotheque: Marker based structured light and high dynamic range imaging for 3d reconstruction. Seminar project, Institute for Computer Graphics and Vision, Graz University of Technology, 2009.

[161] Brad Myers, Scott E. Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7:3–28, March 2000.

[162] Brad A. Myers. A taxonomy of window manager user interfaces. *IEEE Comput. Graph. Appl.*, 8(5):65–84, 1988.

[163] Brad A. Myers. User interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 2:64–103, March 1995.

[164] Brad A. Myers, Choon Hong Peck, Jeffrey Nichols, Dave Kong, and Robert Miller. Interacting at a distance using semantic snarfing. In *Proc. UbiComp 2001*, pages 305–314. Springer-Verlag, 2001.

[165] Miguel Nacenta. Computer vision approaches to solve the screen pose acquisition problem for perspective cursor. Technical Report HCI-TR-06-01, 2006.

[166] Miguel Nacenta, Carl Gutwin, Dzmitry Aliakseyeu, and Sriram Subramanian. There and back again: Cross-display object movement in multi-display environments. *Journal of Human-Computer Interaction*, 24(1):170–229, 2009.

[167] Miguel A. Nacenta, Dzmitry Aliakseyeu, Sriram Subramanian, and Carl Gutwin. A Comparison of Techniques for Multi-Display Reaching. In *Proc. CHI 2005*, pages 371–380. ACM, 2005.

[168] Miguel A. Nacenta, Regan L. Mandryk, and Carl Gutwin. Targeting across display-less space. In *Proc. CHI 2008*, pages 777–786. ACM, 2008.

[169] Miguel A. Nacenta, Satoshi Sakurai, Tokuo Yamaguchi, Yohei Miki, Yuichi Itoh, Yoshifumi Kitamura, Sriram Subramanian, and Carl Gutwin. E-conic: a Perspective-Aware Interface for Multi-Display Environments. In *Proc. UIST 2007*, pages 279–288. ACM, 2007.

[170] Miguel A. Nacenta, Samer Sallam, Bernard Champoux, Sriram Subramanian, and Carl Gutwin. Perspective Cursor: Perspective-Based Interaction for Multi-Display Environments. In *Proc. CHI 2006*, pages 289–298. ACM, 2006.

[171] Neslo Software. Desktop rover. URL: http://www.neslosoftware.com/desktopRover.html, Last access: September 2009.

[172] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26:756–777, June 2004.

[173] Chris North and Ben Shneiderman. A taxonomy of multiple window coordinations. Technical report, Univ. of Maryland, College Park, Computer Science Dept., 1997.

[174] Chris North and Ben Shneiderman. Snap-together visualization: a user interface for coordinating visualizations via relational schemata. In *Proc. AVI 2000*, pages 128–135. ACM, 2000.

[175] Nuria Oliver, Greg Smith, Chintan Thakkar, and Arun C. Surendran. Swish: semantic analysis of window titles and switching history. In *Proc. IUI 2006*, pages 194–201. ACM, 2006.

[176] Kyoung S. Park, Abhinav Kapoor, and Jason Leigh. Lessons learned from employing multiple perspectives in a collaborative virtual environment for visualizing scientific data. In *Proc. CVE 2000*, pages 73–82. ACM, 2000.

[177] Mark Perry and Kenton O'Hara. Display-based activity in the workplace. In *Proc. INTERACT 2003*, pages 591–598, 2003.

[178] David Pinelle, Mutasem Barjawi, Miguel Nacenta, and Regan Mandryk. An evaluation of coordination techniques for protecting objects and territories in tabletop groupware. In *Proc. CHI 2009*, pages 2129–2138. ACM, 2009.

[179] David Pinelle, Carl Gutwin, and Saul Greenberg. Task analysis for groupware usability evaluation: Modeling shared-workspace tasks with the mechanics of collaboration. *ACM Trans. Comput.-Hum. Interact.*, 10(4):281–311, 2003.

[180] Christian Pirchheim, Markus Murschitz, Manuela Waldner, and Dieter Schmalstieg. Deskotheque display system. In: The Trend is Your Friend, Sylvia Eckermann and Gerald Nestler, MedienKunstLabor Kunsthaus Graz, September 2009.

[181] Christian Pirchheim, Manuela Waldner, and Dieter Schmalstieg. Deskotheque: Improved spatial awareness in multi-display environments. In *Proc. VR 2009*, pages 123–126, 2009.

[182] Peter Pirolli and Stuart Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *2005 International Conference on Intelligence Analysis*, pages 1–6, 2005.

[183] Matthew D. Plumlee and Colin Ware. Zooming versus multiple window interfaces: Cognitive costs of visual comparisons. *ACM Trans. Comput.-Hum. Interact.*, 13:179–209, June 2006.

[184] Shankar R. Ponnekanti, Brad Johanson, Emre Kiciman, and Armando Fox. Portability, extensibility and robustness in iros. In *Proc. PERCOM 2003*, page 11. IEEE Computer Society, 2003.

[185] Thorsten Prante, Carsten Magerkurth, and Norbert Streitz. Developing cscw tools for idea finding -: empirical results and implications for design. In *Proc. CSCW 2002*, pages 106–115. ACM, 2002.

[186] Thorsten Prante, Norbert Streitz, and Peter Tandler. Roomware: Computers Disappear and Interaction Evolves. *Computer*, 37(12):47–54, 2004.

[187] Patrick Quirk, Tyler Johnson, Rick Skarbez, Herman Towles, Florian Gyarfas, and Henry Fuchs. Ransac-assisted display model reconstruction for projective display. In *Proc. VR 2006*, pages 318–321. IEEE Computer Society, 2006.

[188] Ramesh Raskar and Paul Beardsley. A self-correcting projector. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:504, 2001.

[189] 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 *Proc. Vis 1999*. IEEE Computer Society, 1999.

[190] Ramesh Raskar, Jeroen van Baar, Paul Beardsley, Thomas Willwacher, Srinivas Rao, and Clifton Forlines. iLamps: geometrically aware and self-configuring projectors. *ACM Transactions on Graphics*, 22(3):809–818, 2003.

[191] Ramesh Raskar, Jeroen van Baar, and Jin Xiang Chai. A Low-Cost Projector Mosaic with Fast Registration. In *Proc. ACCV 2002*, pages 114–119, 2002.

[192] Ramesh Raskar, Greg Welch, Matt Cutts, Adam Lake, Lev Stesin, and Henry Fuchs. The Office of the Future: A Unified Approach to Image-Based Modeling and Spatially Immersive Displays. In *Proc. SIGGRAPH 1998*, pages 179–188. ACM, 1998.

[193] Jun Rekimoto. Pick-and-drop: a direct manipulation technique for multiple computer environments. In *Proc. UIST 1997*, pages 31–39. ACM, 1997.

[194] Jun Rekimoto. Time-machine computing: a time-centric approach for the information environment. In *Proc. UIST 1999*, pages 45–54. ACM, 1999.

[195] Jun Rekimoto and Masanori Saitoh. Augmented Surfaces: A Spatially Continuous Workspace for Hybrid Computing Environments. In *Proc. CHI 1999*, pages 378–385, 1999.

[196] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual Network Computing. *IEEE Internet Computing*, 02(1):33–38, 1998.

[197] Peter Riegler. Spatial registration of tracking targets and screens in a multi-display environment. Bachelor's thesis, Graz University of Technology, 2009.

[198] Meredith Ringel. When one isn't enough: an analysis of virtual desktop usage strategies and their implications for design. In *Ext. Abstracts CHI 2003*, pages 762–763. ACM, 2003.

[199] Jonathan C. Roberts. State of the art: Coordinated & multiple views in exploratory visualization. In *Proc. CMV 2007*, pages 61–71. IEEE Computer Society, 2007.

[200] George Robertson, Mary Czerwinski, Patrick Baudisch, Brian Meyers, Daniel Robbins, Greg Smith, and Desney Tan. The Large-Display User Experience. *IEEE Computer Graphics and Applications*, 25(4):44–51, 2005.

[201] George Robertson, Eric Horvitz, Mary Czerwinski, Patrick Baudisch, Dugald Ralph Hutchings, Brian Meyers, Daniel Robbins, and Greg Smith. Scalable fabric: flexible task management. In *Proc. AVI 2004*, pages 85–89. ACM, 2004.

[202] George Robertson, Maarten van Dantzich, Daniel Robbins, Mary Czerwinski, Ken Hinckley, Kirsten Risden, David Thiel, and Vadim Gorokhovsky. The Task Gallery: A 3D Window Manager. In *Proc. CHI 2000*, pages 494–501. ACM, 2000.

[203] George G. Robertson and Jock D. Mackinlay. The document lens. In *Proc. UIST 1993*, pages 101–108. ACM, 1993.

[204] Yvonne Rogers and Sian E. Lindley. Collaborating around vertical and horizontal large interactive displays: which way is best? *Interacting with Computers*, 16(6):1133–1152, 2004.

[205] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: a middleware platform for active spaces. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):65–67, 2002.

[206] Daniel M. Russell, Mark J. Stefik, Peter Pirolli, and Stuart K. Card. The cost structure of sensemaking. In *Proc. CHI 1993*, pages 269–276. ACM, 1993.

[207] Satoshi Sakurai, Yuichi Itoh, Yoshifumi Kitamura, Miguel A. Nacenta, Tokuo Yamaguchi, Sriram Subramanian, and Fumio Kishino. Interactive systems. design, specification, and verification. chapter A Middleware for Seamless Use of Multiple Displays, pages 252–266. Springer-Verlag, Berlin, Heidelberg, 2008.

[208] Tony Salvador, Jean Scholtz, and James Larson. The denver model for groupware design. *SIGCHI Bull.*, 28:52–58, January 1996.

[209] Joaquim Salvi, Jordi Pagès, and Joan Batlle. Pattern codification strategies in structured light systems. *Pattern Recognition*, 37:827–849, 2004.

[210] Johan Sanneblad and Lars Erik Holmquist. Ubiquitous graphics: combining hand-held and wall-size displays to interact with large images. In *Proc. AVI 2006*, pages 373–377. ACM, 2006.

[211] Robert W. Scheifler and Jim Gettys. The x window system. *ACM Trans. Graph.*, 5:79–109, April 1986.

[212] Chris Schoeneman, Ryan Breen, Guido Poschta, Bertrand Landry Hetu, Tom Chadwick, and Brent Priddy. Synergy. URL: http://synergy2.sourceforge.net/, Last access: September 2009.

[213] Stacey D. Scott, M. Sheelagh T. Carpendale, and Stefan Habelski. Storage bins: Mobile storage for collaborative tabletop displays. *IEEE Comput. Graph. Appl.*, 25:58–65, July 2005.

[214] Stacey D. Scott, Karen D. Grant, and Regan L. Mandryk. System guidelines for co-located, collaborative work on a tabletop display. In *Proc. ECSCW 2003*, pages 159–178. Kluwer Academic Publishers, 2003.

[215] Stacey D. Scott, M. Sheelagh, T. Carpendale, and Kori M. Inkpen. Territoriality in collaborative tabletop workspaces. In *Proc. CSCW 2004*, pages 294–303. ACM, 2004.

[216] CE Shannon and W Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Illinois, 1949.

[217] Chia Shen, Alan Esenther, Clifton Forlines, and Kathy Ryall. Three modes of multi-surface interaction and visualization. In *Information Visualization and Interaction Techniques for Collaboration across Multiple Displays Workshop associated with CHI'06 International Conference*, 2006.

[218] Ben Shneiderman and Aleks Aris. Network visualization by semantic substrates. *IEEE TVCG*, 12(5):733–740, 2006.

[219] Garth B. D. Shoemaker and Kori M. Inkpen. Single display privacyware: augmenting public displays with private information. In *Proc. CHI 2001*, pages 522–529. ACM, 2001.

[220] Lauren Shupp, Robert Ball, Beth Yost, John Booker, and Chris North. Evaluation of viewport size and curvature of large, high-resolution displays. In *Proc. GI 2006*, pages 123–130. Canadian Information Processing Society, 2006.

[221] Ethan Solomita, James Kempf, and Dan Duchamp. XMOVE: A Pseudoserver for X Window Movement. *The X Resource*, 11:143–170, 1994.

[222] Martin Spindler, Sophie Stellmach, and Raimund Dachselt. Paperlens: advanced magic lens interaction above the tabletop. In *Proc. ITS 2009*, pages 69–76. ACM, 2009.

[223] Martin Spindler, Christian Tominski, Heidrun Schumann, and Raimund Dachselt. Tangible views for information visualization. In *Proc. ITS 2010*. ACM Press, 2010.

[224] Oliver G. Staadt, Benjamin A. Ahlborn, Oliver Kreylos, and Bernd Hamann. A foveal inset for large display environments. In *Proc. VRCAI 2006*, pages 281–288. ACM, 2006.

[225] Mark Stefik, Gregg Foster, Daniel G. Bobrow, Kenneth Kahn, Stan Lanning, and Lucy Suchman. Beyond the chalkboard: computer support for collaboration and problem solving in meetings. *Commun. ACM*, 30(1):32–47, 1987.

[226] Jason Stewart, Benjamin B. Bederson, and Allison Druin. Single display groupware: a model for co-present collaboration. In *Proc. CHI 1999*, pages 286–293. ACM, 1999.

[227] Richard Stoakley, Matthew J. Conway, and Randy Pausch. Virtual reality on a wim: interactive worlds in miniature. In *Proc. CHI 1995*, pages 265–272. ACM Press/Addison-Wesley Publishing Co., 1995.

[228] Maureen C. Stone, Ken Fishkin, and Eric A. Bier. The movable filter as a user interface tool. In *Proc. CHI 1994*, pages 306–312. ACM, 1994.

[229] Marc Streit, Hans-Jörg Schulz, Dieter Schmalstieg, and Heidrun Schumann. Towards multi-user multi-level interaction. In *Proc. CoVIS Workshop 2009*, 2009.

[230] Norbert A. Streitz, Jörg Geissler, Torsten Holmer, Shin'ichi Konomi, Christian Müller-Tomfelde, Wolfgang Reischl, Petra Rexroth, Peter Seitz, and Ralf Steinmetz. i-LAND: An interactive Landscape for Creativity and Innovation. In *Proc. CHI 1999*, pages 120–127. ACM, 1999.

[231] Stefan Stumpfl. Single user projected displays. Bachelor's thesis, Graz University of Technology, 2009.

[232] Wolfgang Stürzlinger, Olivier Chapuis, Dusty Phillips, and Nicolas Roussel. User Interface Façades: Towards Fully Adaptable User Interfaces. In *Proc. UIST 2006*, pages 309–318, 2006.

[233] Ramona Su and Brian P. Bailey. Put them where? towards guidelines for positioning large displays in interactive workspaces. In *Proc. INTERACT 2005*, pages 337–349, 2005.

[234] Kishore Swaminathan and Steve Sato. Interaction design for large displays. *Interactions*, 4(1):15–24, 1997.

[235] Susanne Tak and Andy Cockburn. Improved window switching interfaces. In *Ext. Abstracts CHI 2010*, pages 2915–2918. ACM, 2010.

[236] Susanne Tak, Andy Cockburn, Keith Humm, David Ahlström, Carl Gutwin, and Joey Scarr. Improving window switching interfaces. In *Proc. INTERACT 2009*, pages 187–200, 2009.

[237] Desney S. Tan and Mary Czerwinski. Effects of visual separation and physical discontinuities when distributing information across multiple displays. In *Proc. INTERACT 2003*, pages 252–255, 2003.

[238] Desney S. Tan, Brian Meyers, and Mary Czerwinski. Wincuts: manipulating arbitrary window regions for more effective use of screen space. In *Ext. Abstracts CHI 2004*, pages 1525–1528. ACM, 2004.

[239] Peter Tandler, Thorsten Prante, Christian Müller-Tomfelde, Norbert Streitz, and Ralf Steinmetz. Connectables: dynamic coupling of displays for the flexible creation of shared workspaces. In *Proc. UIST 2001*, pages 11–20. ACM, 2001.

[240] Anthony Tang, Melanie Tory, Barry Po, Petra Neumann, and Sheelagh Carpendale. Collaborative coupling over tabletop displays. In *Proc. CHI 2006*, pages 1181–1190. ACM, 2006.

[241] Masayuki Tani, Masato Horita, Kimiya Yamaashi, Koichiro Tanikoshi, and Masayasu Futakawa. Courtyard: integrating shared overview on a large screen and per-user detail on individual screens. In *Proc. CHI 1994*, pages 44–50. ACM, 1994.

[242] Craig Tashman. Windowscape: a task oriented window manager. In *Proc. UIST 2006*, pages 77–80. ACM, 2006.

[243] Matthew Tobiasz, Petra Isenberg, and Sheelagh Carpendale. Lark: Coordinating co-located collaboration with information visualization. *IEEE TVCG*, 15(6):1065–1072, 2009.

[244] Theophanis Tsandilas and Ravin Balakrishnan. An evaluation of techniques for reducing spatial interference in single display groupware. In *Proc. ECSCW 2005*, pages 225–245. Springer-Verlag New York, Inc., 2005.

[245] Edward Tse, Jonathan Histon, Stacey D. Scott, and Saul Greenberg. Avoiding interference: how people use spatial separation and partitioning in sdg workspaces. In *Proc. CSCW 2004*, pages 252–261. ACM, 2004.

[246] Philip Tuddenham, Ian Davies, and Peter Robinson. Websurface: an interface for co-located collaborative information gathering. In *Proc. ITS 2009*, pages 181–188. ACM, 2009.

[247] Brygg Ullmer and Hiroshi Ishii. The metadesk: models and prototypes for tangible user interfaces. In *Proc. UIST 1997*, pages 223–232. ACM, 1997.

[248] Andries van Dam. Post-wimp user interfaces. *Commun. ACM*, 40:63–67, February 1997.

[249] Maarten van Dantzich, Vadim Gorokhovsky, and George Robertson. Application Redirection: Hosting Windows Applications in 3D. In *Proceedings of the 1999 workshop on new paradigms in information visualization and manipulation*, pages 87–91, 1999.

[250] Ivan Viola, Armin Kanitsar, and Meister Eduard Groller. Importance-driven volume rendering. In *Proc. VIS 2004*, pages 139–146. IEEE Computer Society, 2004.

[251] Daniel Vogel and Ravin Balakrishnan. Occlusion-aware interfaces. In *Proc. CHI 2010*, pages 263–272. ACM, 2010.

[252] Daniel Wagner and Dieter Schmalstieg. Artoolkitplus for pose tracking on mobile devices. In *Proc. CVWW 2007*, 2007.

[253] Grant Wallace, Otto J. Anshus, Peng Bi, Han Chen, Yuqun Chen, Douglas Clark, Perry Cook, Adam Finkelstein, Thomas Funkhouser, Anoop Gupta, Matthew Hibbs, Kai Li, Zhiyan Liu, Rudrajit Samanta, Rahul Sukthankar, and Olga Troyanskaya.

Tools and Applications for Large-Scale Display Walls. *IEEE Computer Graphics and Applications*, 25:24–33, 2005.

[254] Grant Wallace, Peng Bi, Kai Li, and Otto Anshus. A Multi-Cursor X Window Manager Supporting Control Room Collaboration. Technical Report TR-707-04, Computer Science, Princeton University, 2004.

[255] Grant Wallace, Han Cheny, and Kai Li. Automatic alignment of tiled displays for a desktop environment. *Journal of Software*, 15(12):1776–1786, December 2004.

[256] Grant Wallace and Kai Li. Virtually shared displays and user input devices. In *Proc. USENIX 2007*, pages 1–6, 2007.

[257] James R. Wallace, Regan L. Mandryk, and Kori M. Inkpen. Comparing content and input redirection in mdes. In *Proc. CSCW 2008*, pages 157–166. ACM, 2008.

[258] James R. Wallace, Stacey D. Scott, Taryn Stutz, Tricia Enns, and Kori Inkpen. Investigating teamwork and taskwork in single- and multi-display groupware systems. *Personal Ubiquitous Comput.*, 13(8):569–581, 2009.

[259] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. Guidelines for using multiple views in information visualization. In *Proc. AVI 2000*, pages 110–119. ACM, 2000.

[260] Colin Ware. *Information visualization: perception for design*. Morgan Kaufmann, 2000.

[261] Chris Weaver. Building highly-coordinated visualizations in improvise. In *Proc. InfoVIS 2004*, pages 159–166. IEEE Computer Society, 2004.

[262] Mark Weiser. Some computer science issues in ubiquitous computing. *Commun. ACM*, 36:75–84, July 1993.

[263] Daniel Wigdor and Ravin Balakrishnan. Empirical investigation into the effect of orientation on text readability in tabletop displays. In *Proc. ECSCW 2005*, pages 205–224. Springer-Verlag New York, Inc., 2005.

[264] Daniel Wigdor, Hao Jiang, Clifton Forlines, Michelle Borkin, and Chia Shen. Wespace: the design development and deployment of a walk-up and share multi-surface visual collaboration system. In *Proc. CHI 2009*, pages 1237–1246. ACM, 2009.

[265] Daniel Wigdor, Chia Shen, Clifton Forlines, and Ravin Balakrishnan. Table-centric interactive spaces for real-time collaboration. In *Proc. AVI 2006*, pages 103–107. ACM, 2006.

[266] Ruigang Yang, David Gotz, Justin Hensley, Herman Towles, and Michael S. Brown. PixelFlex: A Reconfigurable Multi-Projector Display System. In *Proc. VIS 2001*, pages 167–174. IEEE Computer Society, 2001.

[267] Jiří Zahrádka. Augmented user interface. Technical report, Graz University of Technology, Brno University of Technology, 2011.

[268] Ana Zanella and Saul Greenberg. Avoiding interference through translucent interface components in single display groupware. In *Proc. CHI 2001*, pages 375–376. ACM, 2001.

[269] Polle T. Zellweger, Jock D. Mackinlay, Lance Good, Mark Stefik, and Patrick Baudisch. City lights: contextual views in minimal space. In *Ext. Abstracts CHI 2003*, pages 838–839. ACM, 2003.

[270] Shumin Zhai, Julie Wright, Ted Selker, and Sabra-Anne Kelin. Graphical means of directing user's attention in the visual interface. In *Proc. INTERACT 1997*, pages 59–66. Chapman & Hall, 1997.