Wideband Displays: Mitigating Multiple Monitor Seams

Jock D. Mackinlay[1]
[1]Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
{mackinlay, jheer}@parc.com

Jeffrey Heer[1,2]
[2]Group for User Interface Research
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720-1776

ABSTRACT
Wideband displays fill our field of view, creating new opportunities to develop effective visual interfaces. Although multiple monitors are becoming an affordable way to create wideband displays, the resulting seams create gaps in words and divide diagonal lines into nonaligned segments. We present several novel user interface techniques for creating seam-aware applications, showing that vendors need not wait for affordable seamless displays to exploit the potential of wideband displays.

Categories & Subject Descriptors: H5.2 [**Information interfaces and presentation (e.g., HCI)**] User Interfaces – *Graphical user interfaces (GUI), Screen design (e.g. text, graphics, color), Windowing systems*.

General Terms: Design, Human Factors**.**

Keywords: Seam-aware interfaces, wideband displays, multiple monitors.

INTRODUCTION
Progress in information visualization and our understanding of human-information interaction provides opportunities to develop wideband visual interfaces that leverage displays that fill our field of view. These could radically improve productivity in many knowledge management tasks, analogous to the improved productivity of a craftsman who has the right tools and an ample workbench.

Although multiple monitors are becoming a cost-effective way to create wideband displays, the seams between monitors divide diagonal lines into nonaligned segments and create gaps in words, as shown in Figure 1. The Gestalt Law of Continuity indicates that people do not tend to see nonaligned segments as part of the same line [7]. Words with gaps are also difficult to read.

Even though multiple monitors create seams, they have proven to be effective in niche applications such as CAD and graphic design, particularly when the seams help to organize and align the work [1]. Multiple monitors are also effective when the information for a task fits in windows that are small enough to be placed on individual monitors and those windows do not end up being placed across seams. For example, a stockbroker might assign various windows showing different types of information to different monitors in the morning and not have to worry about the seams for the rest of the day. Grudin has found that multiple monitor use confers many benefits, including peripheral awareness and improved resource access [3]. Furthermore, Czerwinski et al review many related studies that also indicate benefits for increased display size. Their recent study found that a large research display had significant benefits over a standard LCD monitor for complex multi-application computer tasks [1].

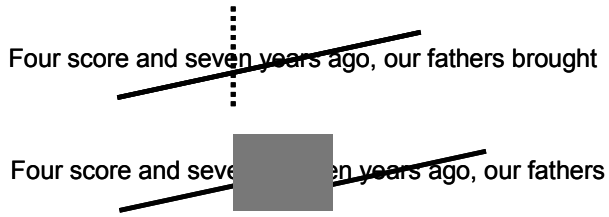

Figure 1: Computer view (top) and human view (bottom). The dotted line indicates a multiple monitor seam. People see broken text and graphics.

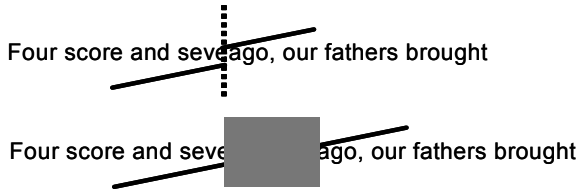

Figure 2: Removing space fixes the break in the line but makes the text appear occluded.

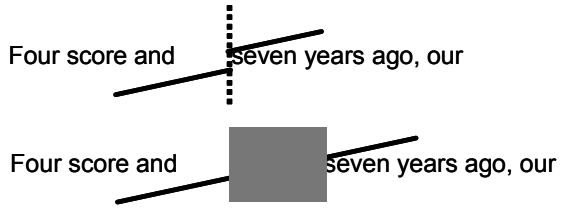

Figure 3: Occlusion can be avoided by moving the text.

However, tasks that involve the frequent creation of windows may require additional support to avoid the overhead of moving windows off seams. For example, nVidia, a major supplier of graphics cards that support multiple monitors, provides a driver for their card that automatically moves windows off seams onto the closest monitor. However, this automatic movement can obscure windows needed for the task or move windows that are not

CHI 2004, April 24–29, 2004, Vienna, Austria.
ACM 1-58113-703-6/04/0004.

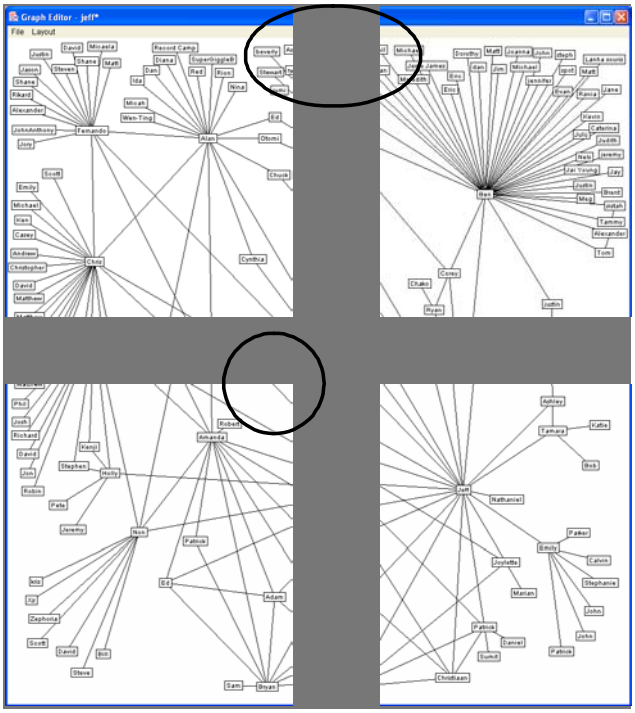

Figure 4: Seam-ignored graph layout. The circle indicates a line segment that does not appear to be part of a link between nodes. The oval indicates split nodes.

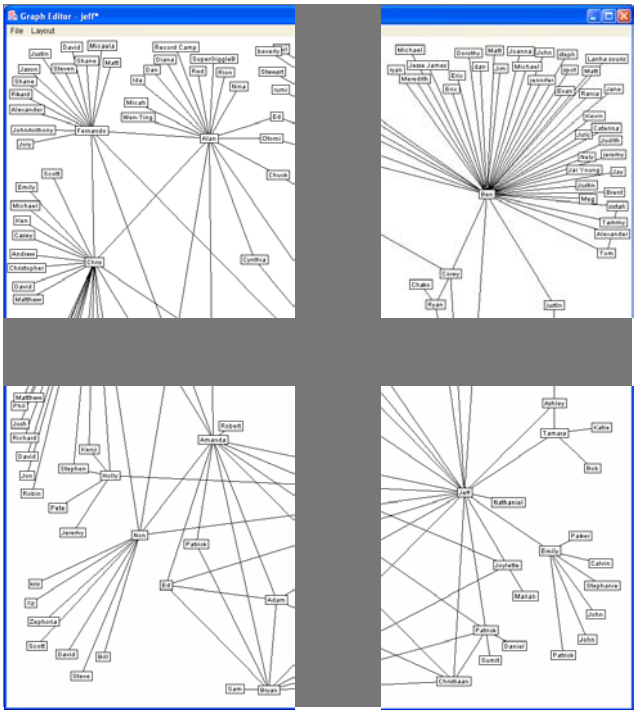

Figure 5: Seam-aware graph layout. Links are drawn under seams, and nodes are moved off of seams.

impacted by seams. Furthermore, some tasks require windows that do not fit in a single monitor, in which case simply moving windows will not be effective.

MITIGATING SEAMS

The basic insight about mitigating seams in wideband displays is to acknowledge that seams have a perceptual impact. The user sees the lower half of Figure 1 rather than the top half. Given this insight, we can develop techniques to create seam-aware applications. For example, Figure 2 shows that a line will appear linear if it is drawn as if there were a display behind the seam. However, text becomes occluded by the seam. Figure 3 shows how text can be moved off of a seam, avoiding both the broken word in Figure 1 and the occlusion in Figure 2.

Two Types of Seam Disruption

A formal analysis of graphical presentations indicates that a seam can disrupt window space in two ways: when the application uses space as *container* to hold graphical objects, or when it uses it as a *metric field* to position objects meaningfully with respect to quantitative axes [6]. In fact, an application can use space simultaneously as a container and as a metric field. For example, charts can have a quantitative axis in one direction and an ordinal or nominal axis in the other direction [6]. Container spaces and metric spaces require different techniques for mitigating seams.

Container spaces and seam-awareness

When an application uses space as a container, it has the freedom to draw *seam-aware* graphics that compensate for the perceptual impact of the seams. Figure 4 is a screen shot of a node/link graph that is not seam aware. Links appear disjoint, and nodes are split across monitors. In contrast, Figure 5 shows the same graph drawn with seam-awareness turned on. Links appear to be drawn through the seams, and nodes are moved off of the seams. The implementation section describes how this can be done interactively.

Metric spaces and City Lights indicators

Metric spaces, on the other hand, must be drawn through a seam to maintain the metric from one monitor to the next, which may cause important information to be occluded. For example, the scatterplot shown in Figure 6 clearly shows the linear trend of the data. The distance between points is meaningful *across* the seam. Maps, which often need to be large, are another example of a metric space that should be maintained across seams.

However, drawing metric spaces through seams can force graphics objects constrained by the metric to be obscured by the seam. For example, the seam obscures many points in the scatterplot shown in Figure 6. When this occurs, City Lights indicators can be used to help the user see that graphical objects are obscured [8]. The implementation section also describes how seam-awareness was added to the third party application that produced Figure 6.

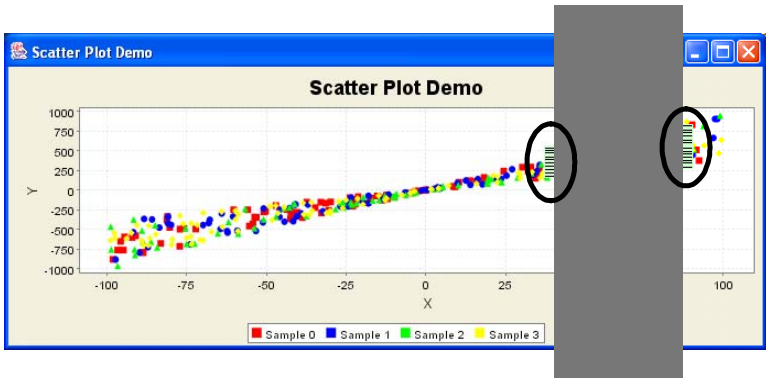

Figure 6: Seam-aware third party scatterplot. Circles enclose City Lights indicators of obscured points [5].

IMPLEMENTATION

Our approach to mitigating seams is to regard the entire display configuration as one large display surface and treat each individual monitor as a viewport into this larger space (as in Figure 7). Physical separations such as seams are explicitly modeled as off-screen pixels in a virtual, seam-aware coordinate space. Naïvely, user interface components can draw themselves onto a virtual canvas and then paint only the visible regions of this canvas to the screen. However, this can result in items being drawn "behind the seams." To make better use of display resources we can make interface components *seam-aware*, structuring their content optimally in the face of possible occlusion by seams. Our technical solution consists of two parts: infrastructural support for computing the seam-aware coordinate spaces of interface components, and software methods for assisting application-specific aspects of seam-awareness. We have implemented our solution in the Java programming language as a general library supporting seam-aware user interfaces built in the Java AWT and Swing user interface toolkits.

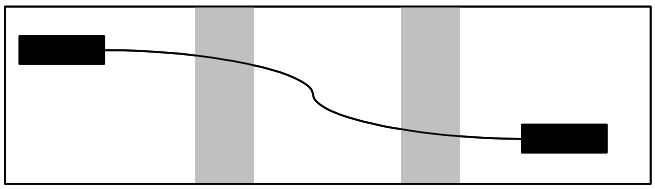

Figure 7: The application is given a graphics object that includes space for the seam. The visible regions are then drawn to the monitors.

Infrastructural Support

At the heart of our infrastructural support is a globally accessible class called the `ScreenGraphFactory`. Upon application launch, the `ScreenGraphFactory` queries the operating system for all the available displays and creates two models of the display set-up. The first model is merely the system of *screen coordinates* provided by the OS. This representation has no knowledge of the actual physical location of monitors, and assumes that adjacent screens form a continuous display. The second model is a *seam-aware coordinate system* that models physical separations (such as seams) as distances measured in pixels.

Successful creation of this seam-aware coordinate model depends on knowing the actual physical distances between display surfaces, represented in terms of display pixels. In initial studies, we manually measured these distances and included them as input parameters. A more general solution is to acquire the seam widths dynamically with a simple calibration application, depicted in Figure 8. For each display boundary, the user performs a simple line fitting task. The system can then use basic trigonometry to infer the correct distance in pixels between the displays. Such calibrations need only be performed once for a given display setup, as the results can be stored and reused.

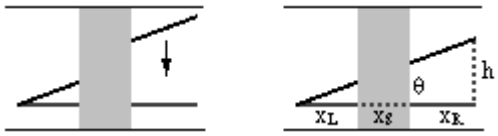

Figure 8: Example of seam width calibration. After the user fits the line on the right side, the seam width x_S can be computed as $x_S = (h / \tan \theta) - (x_L + x_R)$.

The `ScreenGraphFactory` assists user interface components in achieving seam-awareness by providing custom display information on a per component basis. Using the bounding box of the user interface component and the pre-computed seam-aware coordinate space, the `ScreenGraphFactory` constructs a custom seam-aware coordinate space local to the component. This information is encapsulated in a `ScreenGraph`, an object that keeps track of both screen and seam-aware coordinates and methods for mapping between the two. In particular, the `ScreenGraph` abstraction provides methods for retrieving the size of the virtual, seam-aware coordinate space; determining if a point or bounding box is contained within or intersects a seam; and providing the nearest visible display region in any given direction. Seam-aware interface components must query the `ScreenGraphFactory` for `ScreenGraph` instances, and should monitor move and resize events from their parent windows to request updated `ScreenGraph` instances as appropriate.

Application-specific seam management

The `ScreenGraph` allows user interface components to be aware of seams and separations in the display. How the seams are dealt with, however, is left to application-specific code. For example, custom paint routines can draw the component into an offscreen buffer using the seam-aware coordinate space, and then paint only the visible regions to the actual display. Layout algorithms within the application can determine the spatial position of interface items to avoid seam crossings or occlusions. In the next sections we discuss the implementation of multiple classes of seam-aware applications.

Seam-aware Graphics for Container Spaces

As an example of container spaces, we have incorporated seam-awareness into the `prefuse` graph visualization toolkit (http://guir.berkeley.edu/prefuse). To make the toolkit naively seam-aware, we simply modified the display component, which is responsible for drawing visualized graph elements (e.g., nodes and edges) to the screen. The component was modified such that it requests `ScreenGraph` objects from the `ScreenGraphFactory`, and adjusts its offscreen paint buffer to match the size of the virtual coordinate space. The paint routine draws all the graph elements into this buffer, but then only draws to the screen those portions of the buffer that correspond to visible display regions. While this causes all lines and shapes to be rendered correctly, it can cause items to be drawn "behind the seams."

To remedy this problem, we added custom layout procedures to our applications. The `prefuse` architecture employs a modular pipeline architecture, allowing custom processing components to be placed in the pipeline at will. This allowed us to completely avoid rewriting intricate graph layout algorithms, instead adding an additional layout module further down the pipeline that perturbs nodes so that they do not intersect any seam boundaries. Of course, custom layout algorithms for more specialized seam-aware layouts may also be desirable.

Seam-aware Graphics for Metric Spaces

The scatterplot in Figure 6 shows that our architecture can be added to third-party Java applications that require multiple-monitor metric spaces. The scatterplot was drawn after minimal modification to `JFreeChart`, an open source charting package (http://www.jfree.org/jfreechart). The top level paint method was modified to render into an offscreen buffer, which was then mapped to the monitors using a `ScreenGraph` instance.

Adding City Lights indicators to an application requires information about the existence and location of the components in the interface, which can be difficult to determine in third party code. Luckily, `JFreeChart` generates an `EntityCollection` during rendering that contains all relevant components and their locations. The City Lights indicators were added to Figure 6 using a `ScreenGraph` to identify obscured components and find the closest visible point for the indicator.

CONCLUSION

Although researchers are currently developing seamless wideband displays [1,2,5], they are expensive. However, their cost will ultimately become affordable, driven down by computer gaming, entertainment, and teleconferencing. In the meantime, vendors can already explore the potential of wideband visual interfaces by mitigating the seams in multiple-monitor wideband solutions built with current commercial hardware.

In this paper, we describe the differing requirements of container and metric spaces and the implementation of several novel user interface techniques for creating seam-aware applications that target wideband displays based on multiple monitors. Given our practical methods for implementing wideband visual interfaces, the next step is to improve our applications by exploiting the power of the human visual system to work effectively in computational workspaces that are at least as large and high-resolution as the desks on which we work with paper.

ACKNOWLEDGMENTS

This research has been funded in part by contract #MDA904-03-C-0404 awarded to Stuart K. Card and Peter Pirolli from the ARDA Novel Intelligence from Massive Data program. We thank Stuart K. Card for many discussions about window paradigms and Polle T. Zellweger for her skilled suggestions about multiple drafts.

REFERENCES

1. Czerwinski, M., Smith, G., Regan, T., Meyers, B., Robertson, G., Starkweather, G. Toward characterizing the productivity benefits of very large displays. *Proc. Interact 2003*.
2. Baudisch, P., Good, N., Bellotti, V., Schraedley, P. Keeping things in context. *Proc CHI 2002.* 259-266.
3. Grudin, J. Partitioning digital worlds: Focal and peripheral awareness in multiple monitor use. *Proc. CHI 2001*, CHI Letters 3 (1) 458-465.
4. Henderson, D. A., and Card, S. K. Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Transactions on Graphics, 5*, (1986) 211-243.
5. Li. K. et. al. Early experiences and challenges in building and using scalable display wall system. *IEEE CG&A* 20(4) (2000) 671-680.
6. Mackinlay, J. Automating the design of graphical presentation of relational information, *ACM TOG 5 (2)* ACM Press (1986) 110-141.
7. Ware, C. Information Visualization: Perception for Design. Morgan Kaugman (200) 203-213.
8. Zellweger, P., Mackinlay, J., Good, L., Stefik, M., and Baudisch, P. City Lights: Contextual views in minimal space. *Ext. Abstracts CHI2003*, ACM Press (2003) 838-839.