Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics

Benjamin B. Bederson
Bell Communications Research
445 South Street - MRE 2D-336
Morristown, NJ 07960
(bederson@bellcore.com)

James D. Hollan
Computer Science Department
University of New Mexico
Albuquerque, NM 87131
(hollan@cs.unm.edu)

KEYWORDS
Interactive user interfaces, multiscale interfaces, authoring, information navigation, hypertext, information visualization, information physics.

ABSTRACT
We describe the current status of Pad++, an infinite resolution sketchpad that we are exploring as an alternative to traditional window and icon-based approaches to interface design. We discuss the motivation for Pad++, describe the implementation, and present prototype applications. In addition, we introduce an informational physics strategy for interface design and briefly compare it with metaphor-based design strategies.

INTRODUCTION
If interface designers are to move beyond windows, icons, menus, and pointers to explore a larger space of interface possibilities, additional ways of thinking about interfaces that go beyond the desktop metaphor are required. The exploration of virtual 3D worlds is one alternative. It follows quite naturally from more traditional direct manipulation approaches to interface design and involves similar underlying metaphors, although they are enriched by the greater representational possibilities afforded by moving from the desktop to richer 3D worlds.

There are numerous benefits to metaphor-based approaches, but they also lead designers to employ computation primarily to mimic mechanisms of older media. While there are important cognitive, cultural, and engineering reasons to exploit earlier successful representations, this approach has the potential of underutilizing the mechanisms of new media.

For the last few years we have been exploring a different strategy for interface design to help focus on novel mechanisms enabled by computation rather than on mimicking mechanisms of older media. Informally the strategy consists of viewing interface design as the development of a physics of appearance and behavior for collections of informational objects.

Published in UIST '94

For example, an effective informational physics might arrange for useful representation to be a natural product of normal activity. Consider how this is at times the case for the physics of the world. Some materials record their use and in doing so influence future use in positive ways. Used books crack open at often referenced places. Frequently consulted papers are at the top of piles on our desks. Use dog-ears the corners and stains the surface of index cards and catalogs. All these provide representational cues as a natural product of doing but the physics of older media limit what can be recorded and the ways it can influence future use.

Following an informational physics strategy has lead us to explore history-enriched digital objects [12][13]. Recording on objects (e.g. reports, forms, source-code, manual pages, email, spreadsheets) the interaction events that comprise their use makes it possible on future occasions, when the objects are used again, to display graphical abstractions of the accrued histories as parts of the objects themselves. For example, we depict on source code its copy history so that a developer can see that a particular section of code has been copied and perhaps be led to correct a bug not only in the piece of code being viewed but also in the code from which it was derived.

This informational physics strategy has also lead us to explore new physics for interacting with graphical data. In collaboration with Ken Perlin, we have designed a successor to Pad [18]. This system, Pad++, will be the basis for exploration of novel interfaces for information visualization and browsing in a number of complex information-intensive domains. The system is being designed to operate on platforms ranging from high-end graphics workstations to PDAs and Set-top boxes. Here we describe the motivation behind the Pad++ development, report the status of the current implementation, and present some prototype applications.

MOTIVATION

It is a truism of modern life that there is much more information available than we can readily and effectively access. The situation is further complicated by the fact that we are on the threshold of a vast increase in the availability of information because of new network and computational technologies. It is somewhat paradoxical that while we continuously process massive amounts of perceptual data as we experience the world, we have perceptual access to very little of the information that resides within our computing systems or that is reachable via network connections. In addition, this information, unlike the world around is, is rarely presented in ways that reflect either its rich structure or dynamic character.

We envision a much richer world of dynamic persistent informational entities that operate according to multiple physics specifically designed to provide cognitively facile access. The physics need to be designed to exploit semantic relationships explicit and implicit in information-intensive tasks and in our interaction with these new kinds of computationally-based work materials.

One physics central to Pad++ supports viewing information at multiple scales and attempts to tap into our natural spatial ways of thinking. The information presentation problem addressed is how to provide effective access to a large structure of information on a much smaller display. Furnas [10] explored degree of interest functions to determine the information visible at various distances from a central focal area. There is much to recommend the general approach of providing a central focus area of detail surrounded by a periphery that places the detail in a larger context.

With Pad++ we have moved beyond the simple binary choice of presenting or eliding particular information. We can also determine the scale of the information and, perhaps most importantly, the details of how it is rendered can be based on various semantic and task considerations that we describe below. This provides semantic task-based filtering of information that is similar to the early work at MCC on HITS[14] and the recent work of moveable filters at Xerox [3][19].

The ability to make it easier and more intuitive to find specific information in large dataspaces is one of the central motivations for Pad++. The traditional approach is to filter or recommend a subset of the data, hopefully producing a small enough dataset for the user to effectively navigate. Two examples of work of this nature are latent semantic indexing [6] and a video recommender service based on shared ratings with other viewers [11].

Pad++ is complementary to these filtering approaches in that it is a useful substrate to *structure* information. In concert with recommending mechanisms, Pad++ could be used to layout the rated information in a way to make the most highly rated information largest and most obvious, while placing related but lower rated information nearby and smaller.

DESCRIPTION

Pad++ is a general-purpose substrate for exploring visualizations of graphical data with a zooming interface. While Pad++ is not an application itself, it directly supports creation and manipulation of multiscale graphical objects, and navigation through the object space. It is implemented as a widget for Tcl/Tk [17](described in a later section) which provides a simple mechanism for creating zooming-based applications with an interpreted language. The standard objects that Pad++ supports are colored text, text files, hypertext, graphics, and images.

We have written a simple drawing application using Pad++ that supports interactive drawing and manipulation of objects as well loading of predefined or programmatically created objects. This application produced all the figures depicted in this paper.

The basic user interface for Pad++ uses a three button mouse. The left button is mode dependent. For the drawing application shown in this paper, the left button might select and move objects, draw graphical objects, specify where to enter text, etc. The middle button zooms in and the right button zooms out. For systems with two button mice, we have experimented with various mechanisms for mapping zooming in and out to a single button. Typically, this involves having the first motion of the mouse after the button press determine the direction of the zooming.

Pad++ is a natural substrate for representing abstraction of objects using what we term *semantic zooming*. It is natural to see the details of an object when zoomed in and viewing it up close. When zoomed out, however, instead of simply seeing a scaled down version of the object, it is potentially more effective to see a different representation of it. Perlin [18] described a prototype zooming calendar with this notion. There are two ways to describe this type of object. The first is to have different objects, each of which is visible at different, non-overlapping, zooms. This method is supported with the *-minsize* and *-maxsize* options described in the Tcl/Tk Section. The second, and preferred method, is to describe a procedural object that renders itself differently depending on its viewing size. It is possible to prototype procedural objects with Tcl as described below.

RECENT ADVANCES

Our focus in the current implementation has been to provide smooth zooming in a system that works with very

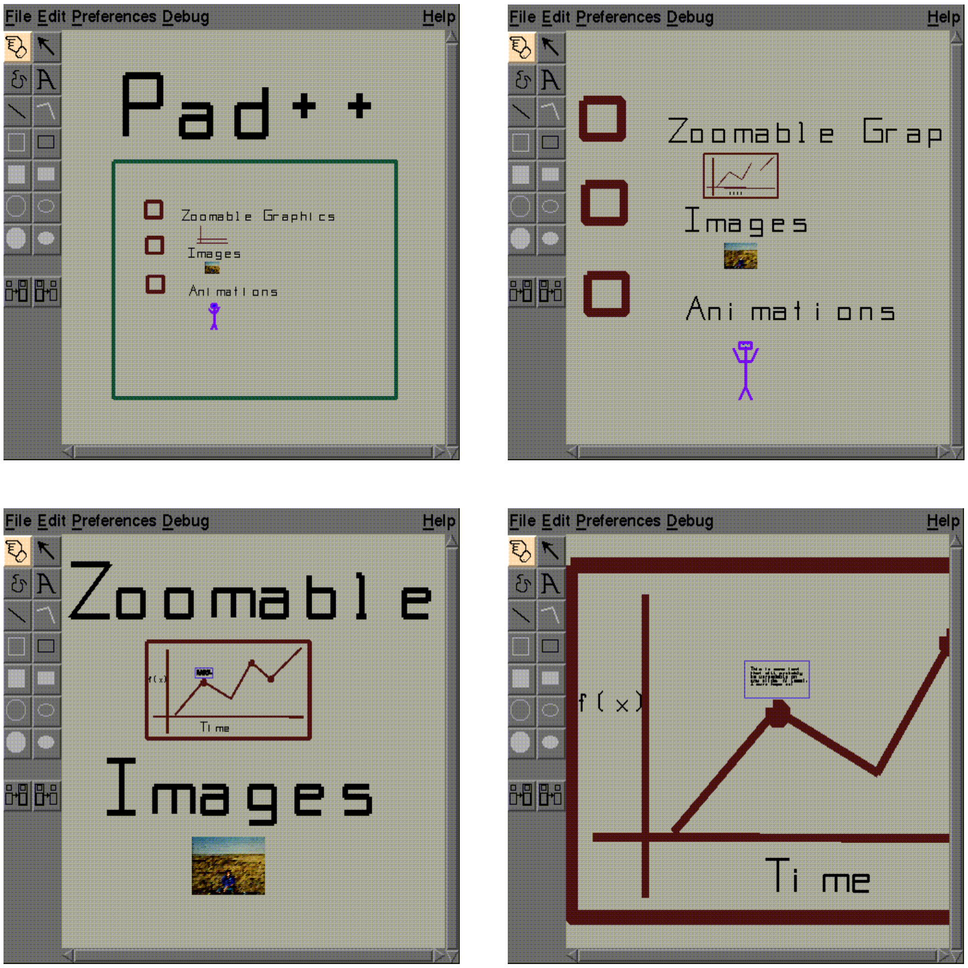

Figure 1: Sequence of snapshots (from left to right and top to bottom) as the view is zoomed in to a hand-drawn picture.

large graphical datasets. The nature of the Pad++ interface requires consistent high frame-rate interactions, even as the dataspace becomes large and the scene gets complicated. In many applications, speed is important, but not critical to functionality. In Pad++, however, the interface paradigm is inherently based on interaction. The searching strategy is to visually explore the dataspace, so it is essential that interactive frame rates be maintained.

IMPLEMENTATION

We implemented Pad++ in C++. There are two versions: one uses Silicon Graphics computers graphics language facilities (GL), and the other uses standard X. The X version runs on SGI's, Suns, PC's running Linux, and should be trivially portable to other standard Unix system. Pad++ is implemented as a widget for Tcl/Tk which allows applications to be written in the interpreted Tcl language. All Pad++ features are accessible through Tcl making it unnecessary to write any new C code.

EFFICIENCY

In order to keep the animation frame-rate up as the dataspace size and complexity increases, we implemented several standard efficiency methods, which taken together create a powerful system. We have successfully loaded over 600,000 objects and maintained interactive rates.

Briefly, the implemented efficiency methods include:

- **Spatial Indexing:** Create a hierarchy of objects based on bounding boxes to quickly index to visible objects.
- **Restructuring:** Automatically restructure the hierarchy of objects to maintain a balanced tree which is necessary for the fastest indexing.

- **Spatial Level-Of-Detail:** Render only the detail needed, do not spend time rendering what can not be seen.
- **Clipping:** Only render the portions of objects that are actually visible.
- **Refinement:** Render fast with low resolution while navigating and refine the image when still.
- **Adaptive Render Scheduling:** Keep the zooming rate constant even as the frame rate changes.

One challenge in navigating through any large dataspace is maintaining a sense of relationship between what you are looking at and where it is with respect to the rest of the data. The rough animation or jumpy zooming as implemented in the original Pad [18] can be disorienting and thus not provide the most effective support for the cognitive and perceptual processing required for interactive information visualization and navigation.

An important interactive interface issue when accessing external information sources is how to give the user access to them without incurring substantial start-up costs while the database is parsed and loaded. In Pad++ this is accomplished with *parallel lazy loading*: only load the portion of the database that is visible in the current view. As the user navigates through the database and looks at new areas, those portions of the database are loaded. This lazy loading is accomplished in the background so the user can continue to interact with Pad++. When the loading is complete, items appear in the appropriate place.

An associated concept is that of ephemeral objects. Objects in Pad++ which are representations of data on disk can be labeled *ephemeral*. These objects are automatically deleted if they have not been viewed in several minutes, thus freeing system resources. When they are viewed again, they are loaded again in parallel as described above.

HYPERTEXT

In traditional window-based systems, there is no graphical depiction of the relationship among windows even when there is a strong semantic relationship. This problem typically comes up with hypertext. In many hypertext systems, clicking on a hyperlink brings up a new window with the linked text (or alternatively replaces the contents of the existing window). While there is an important relationship between these windows (parent and child), this relationship is not represented.

We have begun experimenting with multiscale layouts of hypertext traversals where we graphically represent the parent-child relationships between links. When a hyperlink is selected, the linked data is loaded to the side and made smaller, and the view is animated to center the new data object.

The user interface for accessing hypertext in Pad++ is quite simple. The normal navigation techniques are available, and in addition, clicking on a hyperlink loads in the associated data as described above, and shift-clicking anywhere on a hypertext object animates the view back to that object's parent.

Pad++ can read in hypertext files written in the Hypertext Markup Language (HTML), the language used to describe the well-known hypertext system, MOSAIC (from the NCSA at the University of Illinois). While we do not yet follow links across the network, we can effectively use Pad++ as an alternative viewer to MOSAIC within our file system. Figure 2 shows a snapshot with the MOSAIC home-page loaded and several links followed.

INTERFACE TO TCL/TK

Pad++ is built as a new widget for Tk which provides for simple access to all of its features through Tcl, an interpreted scripting language. Tcl and Tk [17] are an increasingly popular combination of scripting language and Motif-like library for creating graphical user interfaces and applications without writing any C code. The Tcl interface to Pad++ is designed to be very similar to the interface to the Tk Canvas widget - which provides a surface for drawing structured graphics.

While Pad++ does not implement everything in the Tk Canvas yet, it adds many extra features - notably those supporting multiscale objects and zooming. In addition, it supports images, text files, and hypertext, as well as several navigation tools including content-based search. As with the Canvas, Pad++ supports many different types of structured graphics, and new graphical widgets can be added by writing C code. Significantly, all interactions with Pad++ are available through Tcl.

Since Tcl is interpreted and thus slower than compiled code, it is important to understand what its role is in a real-time animation system such as Pad++. There are three classes of things that one can do with Pad++, and the importance of speed varies:

- **Create objects:** Slow - Tcl is fine
- **Handle events:** Medium - Small amount of Tcl is ok
- **Render scene:** Fast - C++ only

Because all rendering is done in C++, and typically only a few lines of Tcl are written to handle each event, Pad++ maintains interactive rates despite its link to Tcl. Tcl is quite good, however, for reading and parsing input files, and creating and laying out graphical multiscale objects.

The Tcl interface to Pad++ is, as previously mentioned, quite similar to that of the Tk canvas, and is summarized

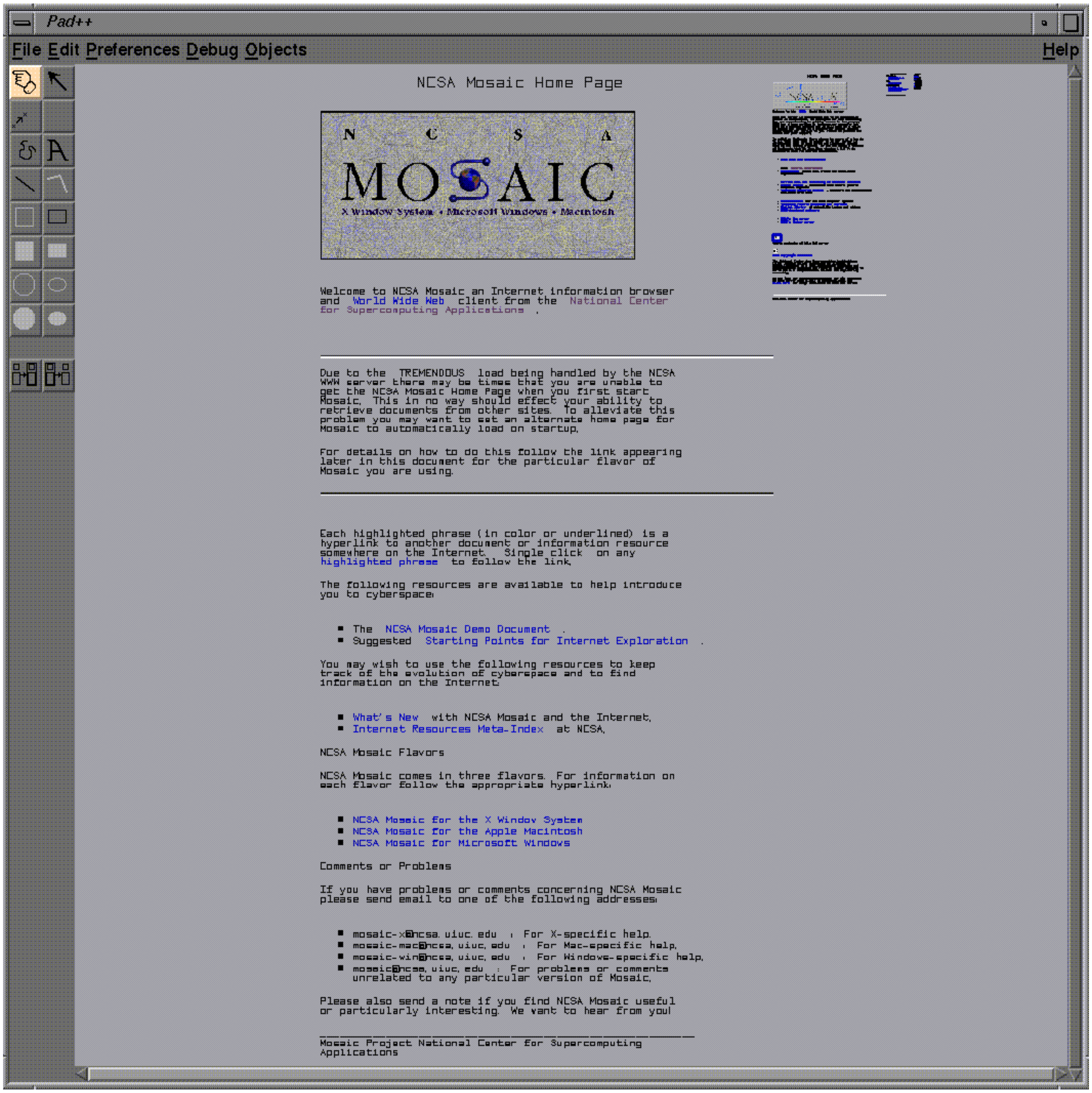

Figure 2: Hypertext. Links are followed and placed on the surface to the side, and made smaller.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.