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, zooming interfaces, authoring, information navigation, hypertext, information visualization, information physics.

ABSTRACT
We describe the current status of Pad++, a zooming graphical interface 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, new interaction techniques must go beyond the desktop metaphor. While several groups are exploring virtual 3D worlds [4][8], we have developed a 2D interface based on zooming. With our system, Pad++, graphical data objects of any size can be created, and zooming is a fundamental interaction technique.

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.

For example, an effective informational physics might arrange for useful representations 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 interaction 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 [11][12]. 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 [17] which is an graphical interface based on zooming. 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

[*] This author has moved to the University of New Mexico, Computer Science Department, Albuquereque, NM 87131, bederson@cs.unm.edu.

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 us, 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 different 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 body of information on a much smaller display. Furnas [9] 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[13] and the recent work of moveable filters at Xerox [3][18].

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 recent examples of work of this nature are latent semantic indexing [5] and a video recommender service based on shared ratings with other viewers [10].

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 [16] (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. Pad++ always zooms around the current cursor position - thus the user can control the zooming dynamically by moving the mouse while zooming. For systems with a two button mouse, 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 [17] described a prototype zooming calendar with this notion. We foresee 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 or other characteristics. 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 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

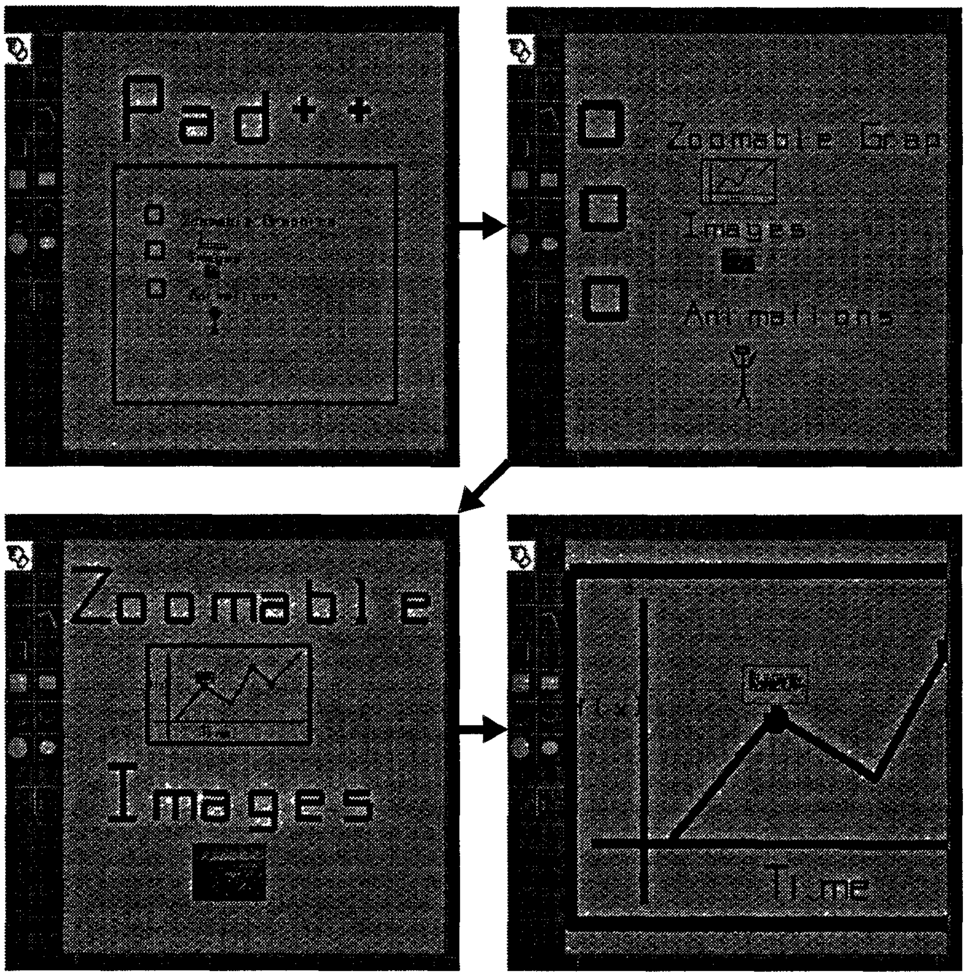


Figure 1: Sequence of snapshots (from left to right and top to bottom) as the view is zoomed in to a hand-drawn picture.

the dataspace, so it is essential that interactive frame rates be maintained.

IMPLEMENTATION

We implemented Pad++ in C++. It runs on either of two graphics systems: the Silicon Graphics computers graphics language facilities (GL); and standard X. The X version runs on SGI's, Suns, PC's running Linux, and should be trivially portable to other standard Unix[R] 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 render what can not be seen.
- **Clipping:** Only render the portions of objects that are actually visible.
- **Refinement:** Render fast with low resolution while navi-

[R]Unix is a registered trademark of Unix Systems Laboratories, Inc.

gating 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 (i.e., balancing local detail and global context). The rough animation or jumpy zooming as implemented in the original Pad [17] 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 (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 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.

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 objects in 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 one of the author's homepage 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 [16] 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:

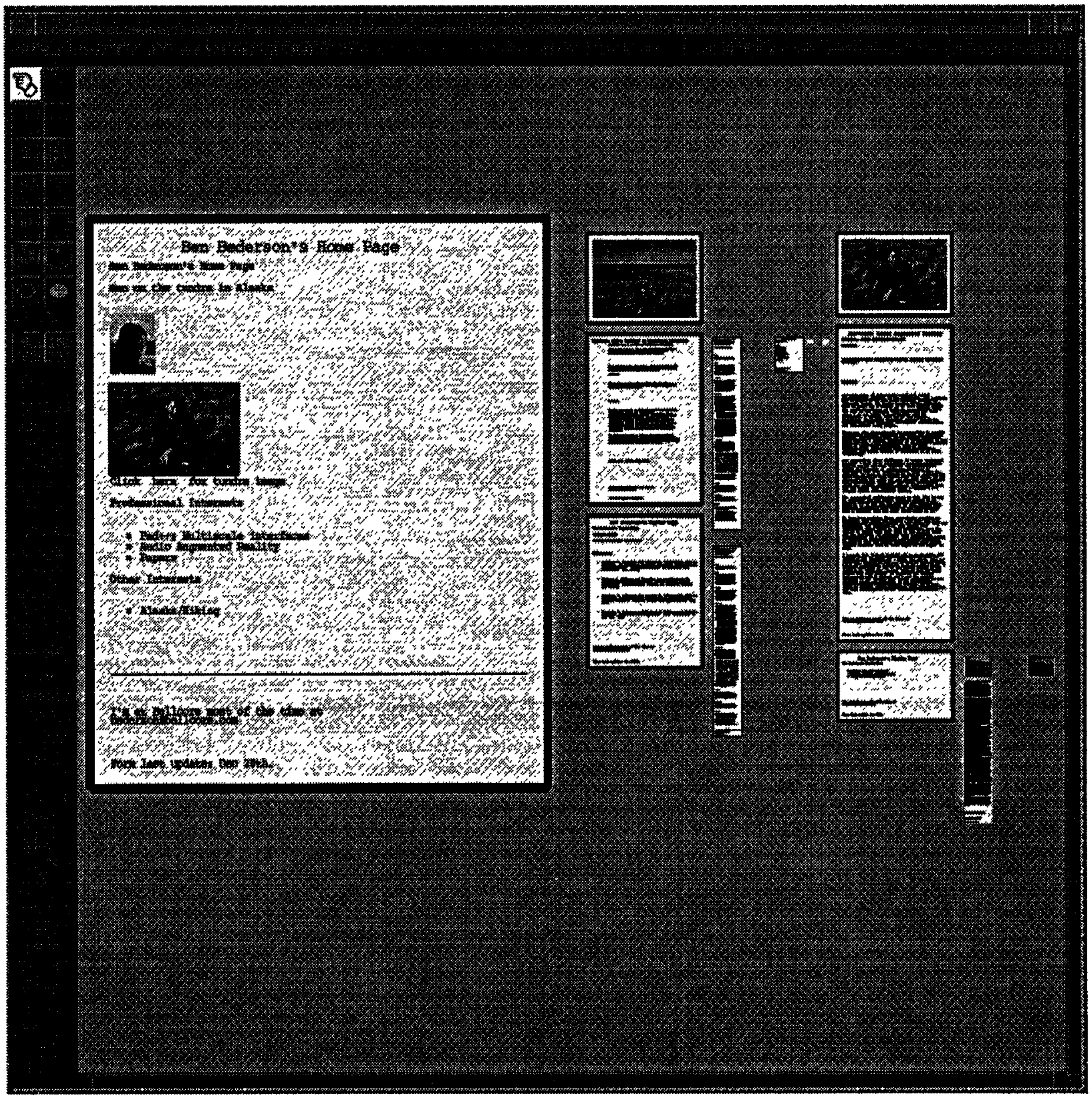
- **Create objects:** Slow - Tcl is fine
- **Handle events:** Medium - Small amount of Tcl is ok
- **Render scene:** Fast - C++ only

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 here to give a feel for what it is like to program Pad++. Every object is assigned a unique integer id. In addition, the user may associate an arbitrary list of text tags with each object. Every command can be directed to either a specific object id or to a tag, in which case it will apply to all objects that share that tag - implicitly grouping objects. Each Pad++ widget has its own name. All Pad++ commands start with the name of the widget, and in the examples that follow, the name of the widget is `.pad`.

Examples:

- A red rectangle with a black outline is created whose corners are at the points (0, 0) and (2, 1):
```
.pad create rectangle 0 0 2 1 -fill red
     -outline black
```
- Put item number 5 at the location (3, 3), make the object



Ben Bederson's Home Page

Ben Bederson's Home Page

Ben on the tundra in Alaska

Click here for tundra image

Professional Interests

- Pad++ Multiscale Interfaces
- Audio Augmented Reality
- Papers

Other Interests

- Alaska/Hiking

I'm at Bellcore most of the time at
bederson@bellcore.com

Page last updated: Dec 20th.

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

twice as big, and make the object anchored at that point on its northwest corner:

```
.pad itemconfig 5 -anchor nw -place "3 3 2"
```

- Specify that item number 5 should only be visible when its largest dimension is greater than 20 pixels and less than 100 pixels.

```
.pad itemconfig 5 -minsize 20 -maxsize 100
```

- Make all items with tag `foo` turn blue when the left button of the mouse is pressed over any of those objects:

```
.pad bind foo <ButtonPress>
     {.pad itemconfig foo -fill blue}
```

As mentioned previously, Pad++ is a natural environment to represent abstraction through semantic zooming. Objects can be represented differently depending on their size by defining *procedural objects*. A procedural object is one that is rendered as a result of a special procedure (as opposed to predefined static objects such as lines or text). Pad++ supports Tcl procedural objects which are very useful for prototyping, but too slow for continued use. Tcl procedural objects work by specifying two Tcl scripts. One returns the bounding box of the object (necessary for efficiency), and the other renders the object (drawing routines are provided). A trivial example is shown here which draws "1993" in red when it is small, and "Jan Feb Mar" in black when it is a little bigger:

```
proc makeCalendar {} {
   .pad create tcl -script "cal" -bb "calBB"
}

proc cal {} {
   set view [.pad move_to]
   set size [lindex $view 2]
   if {$size < .1} {
      .pad set_color red
      .pad set_linewidth 2
      .pad draw_text " 1993" 0 0
   } else {
      .pad set_color black
      .pad set_linewidth 1
      .pad draw_text "Jan Feb Mar" 0 0
   }
}

proc calBB {} {
   return "0 0 11 1"
}
```

NAVIGATION

Finding information on the Pad++ surface is obviously very important as intuitive navigation through large dataspaces is one of its primary motivations. Pad++ support visual searching with zooming in addition to traditional mechanisms, such as content-based search.

Some basic navigation and searching mechanisms are provided at the Tcl interface for the application programmer. A few basic ones are:

- Smoothly go to the location (1, 0) at zoom of 5, and take 1000 milliseconds for the animation:

```
.pad move_to 1 0 5 1000
```

- Smoothly go to the location such that object #37 is centered, and fills three quarter's of the screen, and take 500 milliseconds for the animation:

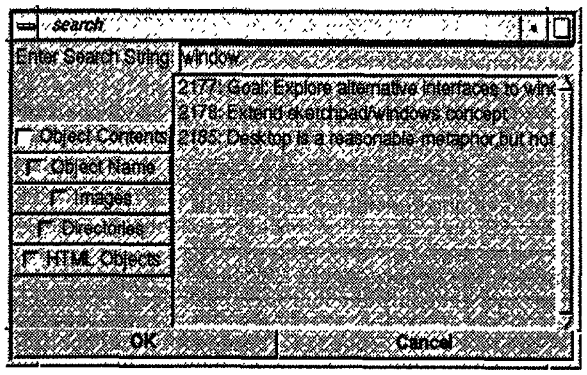
```
.pad center 37 500
```

- Return the list of object ids that contain the text "foo"

```
.pad find withtext foo
```

Figure 3 shows a Tk interface based on these commands. Entering text in the top entry region returns a list of objects that contain that text. Double clicking on any of these objects smoothly animates the view to the specified object.

The smooth animations interpolate in pan and zoom to bring the view to the specified location. If the end point, however, is more than one screen width away from the starting point, the animation zooms out to a point midway between the starting and ending points, far enough out so that both points are visible. The animation then smoothly zooms in to the destination. This gives a sense of context to the viewer as well as speeding up the animation since most of the panning is performed when zoomed out which covers much more ground than panning while zoomed in.



Enter Search String: window

2177: Goal: Explore alternative interfaces to win
2178: Extend sketchpad/windows concept
2185: Desktop is a reasonable metaphor,but not

Figure 3: Content based search

VISUALIZATIONS

We built a Pad++ directory browser to explore how smooth zooming and the various efficiency mechanisms help in viewing a large hierarchical database, and to experiment with multi-scale layouts. The Pad++ directory browser provides a graphical interface for accessing the directory structure of a filesystem (see Figure 4). Each directory is represented by a square frame, and files are represented by solid squares colored by file type. Both directories and files show their filenames as labels when the user is sufficiently close to be able to read them. Each directory has all of its subdirectories and files organized alphabetically inside of it. Searching through the directory structure can be done by zooming in and out of the directory tree, or by using the content based search mechanisms described above. Zooming into a file automatically loads the text inside the colored square and it can then be edited and annotated.

We are able to load in a directory tree with over 600,000 objects, and maintain interactive animation rates of about 10 frames per second. Spatial indexing allows us to explore very

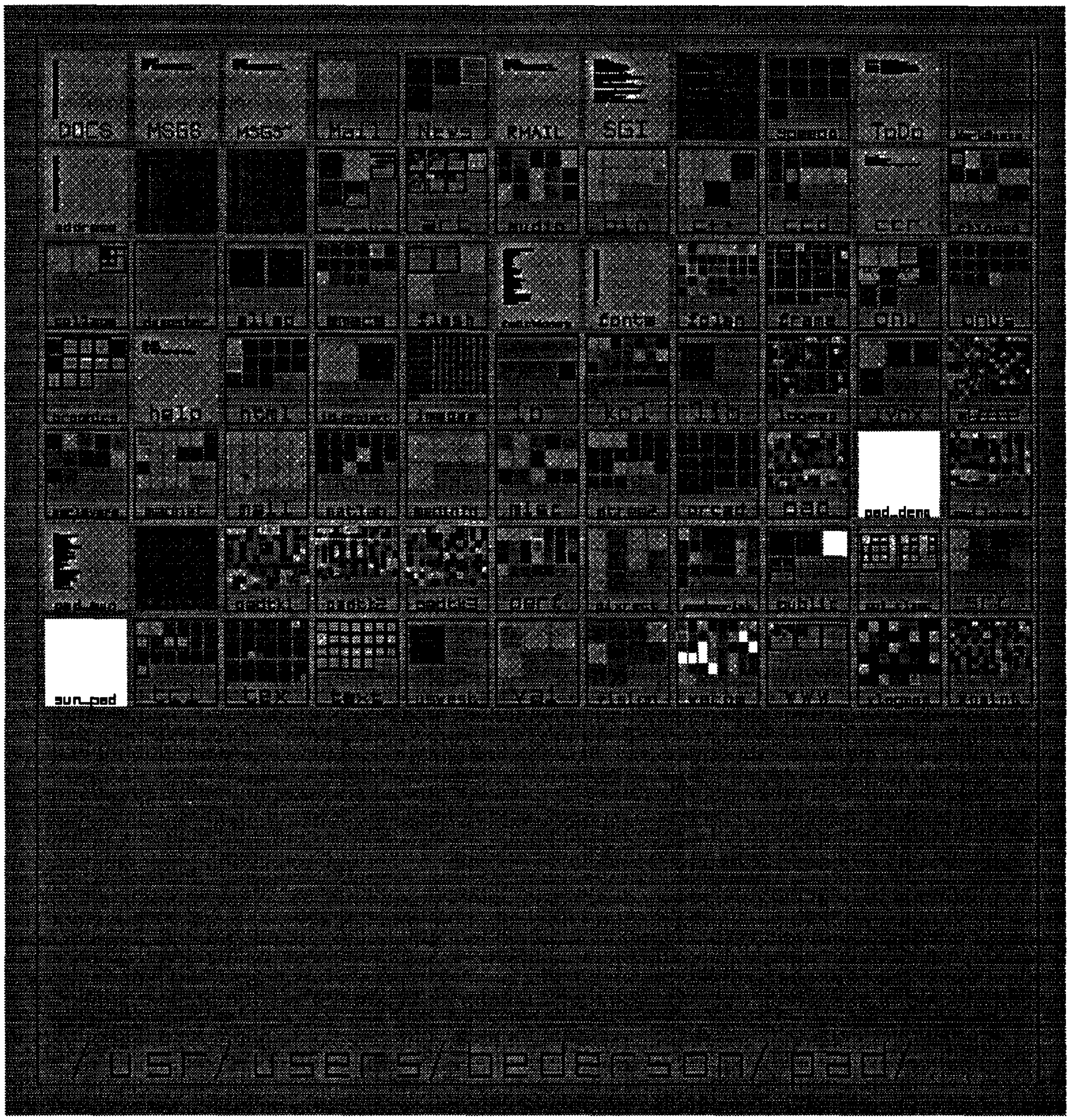


Figure 4: Directory browser snapshot

large databases while keeping the search time fast, since we render only the visible portion of the scene. While navigating with Pad++, very small objects are not drawn and larger ones are drawn with reduced resolution. The objects are then refined and drawn at high resolution when the user stops changing the view.

Another dataset we looked at with Pad++ is a timeline of interesting products, events, and papers in computer technology and user interfaces. History naturally lends itself to being looked back on at different scales. Figure 5 shows a sequence of snapshots as the view is zoomed into the current year. This visualization was created by a Tcl script which reads in a file with a simple format specifying starting and ending dates along with text or images.

PHYSICS AND METAPHOR

As we mentioned earlier, the exploration of Pad++ is part of the development of a more general strategy for interface design. Our goal is to move beyond mimicking the mechanisms of earlier media and start to more fully exploit the radical new mechanisms that computation provides. We think it provides an effective complement to the more traditional metaphor-based approaches. While an informational physics strategy for interface design may certainly involve metaphor, we think there is much that is distinctive about a physics-based perspective. As an interface strategy, it can be distinguished from a metaphor-based strategy in at least four ways.

First, metaphors necessarily pre-exist their use. Pre-Copernicans could never have used the metaphor of the solar system for describing the atom. In designing interfaces, one is limited to the metaphorical resources at hand. In addition, the metaphorical reference must be familiar to work. An unfamiliar interface metaphor is functionally no metaphor at all. One can never design metaphors the way one can design self-consistent physical descriptions of appearance and behavior. Thus, as an interface design strategy, physics offer designability and tailorability that metaphor does not.

Second, metaphors are temporary bridging concepts. When they become ubiquitous, they die. In the same way that linguistic metaphors lose their metaphorical impact (e.g., *foot of the mountain* or *leg of table*), successful metaphors also wind up as dead metaphors (e.g. file, menu, window, desktop). The familiarity provided by the metaphor during earlier stages of use gives way to a familiarity with the interface due to actual experience.

Thus, after awhile, even in the case of metaphor-based interfaces, it is the actual details of appearance and behavior (i.e. the physics) rather than any overarching metaphor that form much of the substantive knowledge of an experienced user. Any restrictions that are imposed on the behaviors of the entities of the interface to avoid violations of the initial metaphor are potential restrictions of functionality that may have been employed to better support the users' tasks and allow the interface to continue to evolve along with the users increasing competency.

The pervasiveness of dead metaphors such as files, menus, and windows may well restrict us from thinking about alternative organizations of computation. New conceptions of persistent objects and organizations of entities into units less monolithic than that of unstructured files are indeed in conflict with older metaphorical notions.

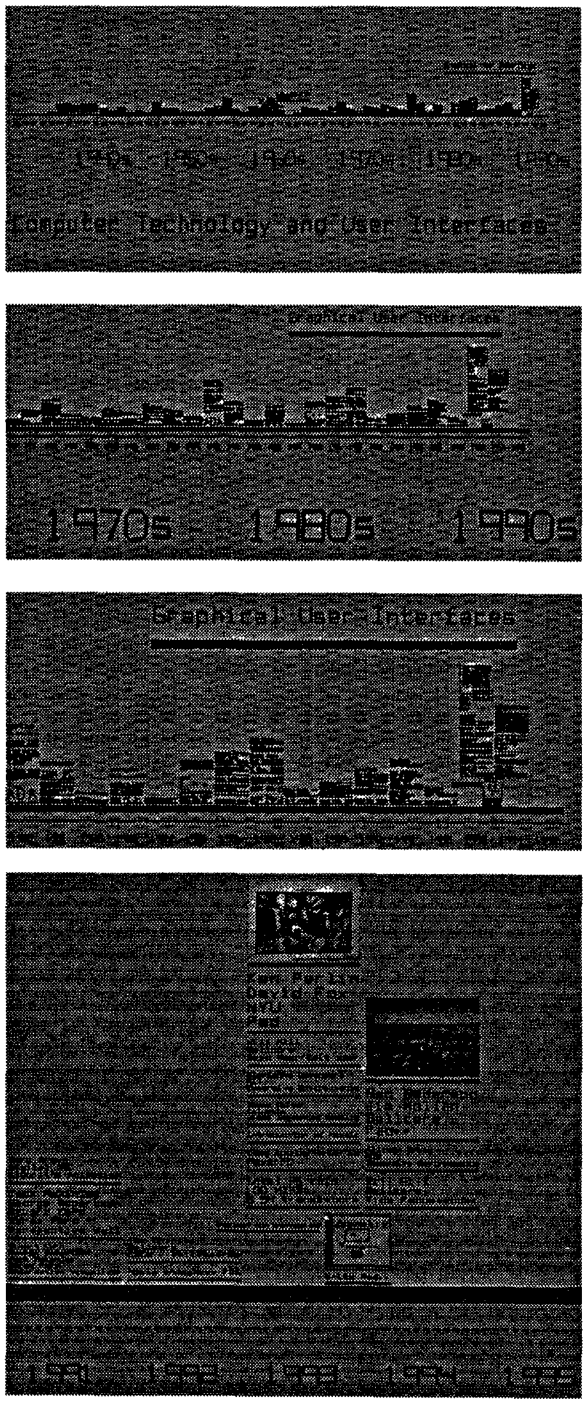

1940s 1950s 1960s 1970s 1980s 1990s
Computer Technology and User Interfaces
Graphical User Interfaces
1970s 1980s 1990s
Graphical User Interfaces
1991 1992 1993 1994 1995

Figure 5: Timeline. Sequence of snapshots (from top to bottom) as the view is zoomed in.

Third, since the sheer amount and complexity of information with which we need to interact continues to grow, we require interface design strategies that *scale*. Metaphor is not such a scaling strategy. Physics is. Physics scales to organize greater and greater complexity by uniform application of sets of simple laws. In contrast, the greater the complexity of the metaphorical reference, the less likely it is that any particular structural correspondence between metaphorical target and reference will be useful.

Fourth, it is clear that metaphors can be harmful as well as helpful since they may well lead users to import knowledge not supported by the interface. There are certainly metaphorical aspects associated with a physics-based strategy. Our point is not that metaphors are not useful but that they may restrict the range of interfaces we consider.

There are, of course, also costs associated in following a physics-based design strategy. One cost is that designers can no longer rely on users' familiarity with the metaphorical reference and this has learnability consequences. However, the power of metaphor comes early in usage and is rapidly superceded by the power of actual experience. Furthermore, since empirical knowability naturally follows from a physics perspective, we can begin to question and quantify how much experimentation will be necessary to learn the designed-in principles, how many inductive steps will be required and of what kinds. Thus, one might want to focus on easily discoverable physics. As is the case with metaphors, all physics are not created equally discoverable or equally fitted to the requirements of human cognition.

CONCLUSION

We implemented Pad++, a zooming graphical interface substrate, focusing on efficiency and expandability. By implementing several efficiency mechanisms which act in concert, we are able to maintain high frame-rate interaction with very large databases. This development is part of an exploration of an informational physics perspective for interface design.

We are currently, in collaboration with NYU, continuing development of the Pad++ substrate as well as starting work in several application domains, such as history-enriched digital objects.

ACKNOWLEDGEMENTS

We would like to thank Ken Perlin and his students, David Fox and Matthew Fuchs, at NYU for enjoyable discussions and for seeding out interest in multiscale interfaces. We especially appreciate the support we have received from Craig Wier as part of ARPA's new HCI Initiative. This will allow us to continue our Pad++ research collaboration with Bellcore and NYU. We also would like to acknowledge other members of the Computer Graphics and Interactive Media Research Group at Bellcore for many discussions shared during our continuing search for the best cheeseburger.

REFERENCES

[1] Ronald M. Baecker, Human Factors and Typography for More Readable Programs, *ACM Press*, 1990.

[2] Benjamin B. Bederson, Larry Stead, and James D. Hollan, Pad++: Advances in Multiscale Interfaces, In *Proceedings of CHI'94 Human Factors in Computing Systems Conference Companion*, ACM/SIGCHI, 1994, pp. 315-316.

[3] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and Magic Lenses: The See-Through Interface, In *Proceedings of 1993 ACM SIGGRAPH Conference*, pp. 73-80.

[4] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. The Information Visualizer, an Information Workspace, In *Proceedings of CHI'91 Human Factors in Computing Systems*, ACM/SIGCHI, 1991, pp. 181-188.

[5] Steve Deerwester, Sue T. Dumais, George W. Furnas, Tom K. Landauer, and Ray Harshman. Indexing by Latent Semantic Analysis. *Journal of American Society of Information Science*, 41, 1990, pp. 391-407.

[6] William C. Donelson. Spatial Management of Information, In Proceedings of *1978 ACM SIGGRAPH Conference*, pp. 203-209.

[7] Stephen G. Eick, Joseph L. Steffen, and Eric E. Sumner, Jr, Seesoft - A Tool for Visualizing Line-Oriented Software Statistics, *IEEE Transactions on Software Engineering*, Vol. 18 (11), pp. 957-968, November, 1992.

[8] Kim M. Fairchild, Steven E. Poltrock, and George W. Furnas. SemNet: Three-Dimensional Graphic Representations of Large Knowledge Bases, in *Cognitive Science and its Applications for Human-Computer Interaction*, Lawrence Erlbaum Associates, 1988.

[9] George W. Furnas, Generalized Fisheye Views, In *Proceedings of CHI'86 Human Factors in Computing Systems*, ACM/SIGCHI, 1986, pp. 16-23.

[10] William C. Hill, videos@bellcore.com: Recommending and evaluating items on the basis of communal history-of-use. *Bellcore Technical Report* #TM-ARH-023560, Morristown, NJ 07960, 1994.

[11] William C. Hill, James D. Hollan, David Wroblewski, and Tim McCandless, Edit Wear and Read Wear, In *Proceedings of CHI'92 Human Factors in Computing Systems*, ACM/SIGCHI, 1992, pp. 3-9.

[12] William C. Hill and James D. Hollan, History-Enriched Digital Objects, in press.

[13] James D. Hollan, Elaine Rich, William Hill, David Wroblewski, Wayne Wilner, Kent Wittenburg, Jonathan Grudin, and Members of the Human Interface Laboratory. An Introduction to HITS: Human Interface Tool Suite, in *Intelligent User Interfaces*, (Sullivan & Tyler, Eds), 1991, pp. 293-337.

[14] James D. Hollan and Scott Stornetta, Beyond Being There, In *Proceedings of CHI'92 Human Factors in Computing Systems*, ACM/SIGCHI, 1992, pp. 119-125. (also appeared as a chapter in Readings in *Groupware and Computer Supported Cooperative Work* (Becker, Ed.), 1993, pp. 842-848.

[15] George Lakoff and Mark Johnson, *Metaphors We Live By*. University of Chicago Press, 1980.

[16] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison Wesley, 1994.

[17] Ken Perlin and David Fox. Pad: An Alternative Approach to the Computer Interface, In *Proceedings of 1993 ACM SIGGRAPH Conference*, pp. 57-64.

[18] Maureen C. Stone, Ken Fishkin, and Eric A. Bier. The Movable Filter as a User Interface Tool, in *Proceedings of CHI'94 Human Factors in Computing Systems*, ACM/SIGCHI, 1994.

[19] Ivan E. Sutherland. Sketchpad: A man-machine graphical communications systems, In *Proceedings of the Spring Joint Computer Conference*, 1963, pp. 329-346, Baltimore, MD: Spartan Books.