Pad

An Alternative Approach to the Computer Interface

Ken Perlin
David Fox

Courant Institute of Mathematical Sciences
New York University
719 Broadway 12th Floor
New York, NY 10003

Abstract

We believe that navigation in information spaces is best supported by tapping into our natural spatial and geographic ways of thinking. To this end, we are developing a new computer interface model called Pad.

The ongoing Pad project uses a spatial metaphor for computer interface design. It provides an intuitive base for the support of such applications as electronic marketplaces, information services, and on-line collaboration. Pad is an infinite two dimensional information plane that is shared among users, much as a network file system is shared. Objects are organized geographically; every object occupies a well defined region on the Pad surface.

For navigation, Pad uses "portals" - magnifying glasses that can peer into and roam over different parts of this single infinite shared desktop; links to specific items are established and broken continually as the portal's view changes. Portals can recursively look onto other portals. This paradigm enables the sort of peripheral activity generally found in real physical working environments. The apparent size of an object to any user determines the amount of detail it presents. Different users can share and view multiple applications while assigning each a desired degree of interaction. Documents can be visually nested and zoomed as they move back and forth between primary and secondary working attention. Things can be *peripherally* accessible.

In this paper we describe the Pad interface. We discuss how to efficiently implement its graphical aspects, and we illustrate some of our initial applications.

1 Introduction

Imagine that the computer screen is a section of wall about the size of a typical bulletin board or whiteboard. Any area of this surface can then be accessed comfortably without leaving one's chair. Imagine further that by applying extraordinarily good eyesight and eye-hand coordination, a user can both read and write as comfortably on any micron wide section of this surface as on any larger section. This would allow the full use of a surface which is several million pixels long and high, on which one can comfortably create, move, read and compare information at many different scales.

The above scenario would, if feasible, put vast quantities of information directly at the user's fingertips. For example, several million pages of text could be fit on the surface by reducing it sufficiently in scale, making any number of on-line information services, encyclopedias, etc., directly available. In practice one would arrange such a work surface hierarchically, to make things easier to find. In a collaborative environment, one could then see the layout (in miniature) of many other collaborators' surfaces at a glance.

The above scenario is impossible because we can't read or write at microscopic scale. Yet the concept is very natural since it mimics the way we continually manage to find things by giving everything a physical *place*. A good approximation to the ideal depicted would be to provide ourselves with some sort of system of 'magic magnifying glasses' through which we can read, write, or create cross-references on an indefinitely enlargeable ('zoomable') surface. This paper describes the Pad interface, which is designed using these principles.

1.1 Overview of the Paper

We begin section one with a brief summary of the basic ideas and components of the Pad Model. We then finish section one with a comparison of Pad to the window/icon paradigm and a summary of prior work. Section two is a description of a typical Pad application, and section three covers the principles of the Pad system. Section four covers several issues in our implementation of Pad, and section five lists some ongoing and future projects. Finally, section six presents our conclusions and acknowledgments.

1.2 Basic Pad Model

The **Pad Surface** is an infinite two dimensional information plane that is shared among users, much as a network file system is shared. It is populated by **Pad Objects**, where we define a Pad Object to be any entity that the user can interact with (examples are: a text file that can be viewed or edited, a clock program, a personal calendar). Pad Objects are organized geographically; every object occupies a well defined region on the Pad surface.

To make themselves visible, Pad Objects can create two types of "ink", **graphics** and **portals**, and place them on the Pad Surface. A graphic is simply any sort of mark such as a bitmap or a vector. Portals are used for navigation, they are like magnifying glasses that can peer into and roam over different parts of the Pad Surface. A portal may have a highly magnified view or a very broad, panoramic view, and this view can be easily changed. The screen itself is just a special "root" portal.

A portal is *not* like a window, which represents a dedicated link between a section of screen and a specific thing (e.g.: a Unix shell in X-Windows or a directory in the Macintosh Finder). A portal is, rather, a view into the single infinite shared desktop; links to specific items are established and broken continually as the portal's view changes. Also, unlike windows, portals can recursively look onto (and into) other portals.

Figure 1 shows a very large financial document on the Pad surface. The small portal at the top of the figure shows an overview of the entire report. The two other portals show successive closeups of portions of the report.

1.3 Object/Portal Interaction

A Pad object may look quite different when seen through different portals. There are two techniques that allow objects vary their appearance: *semantic zooming* and *portal filters*.

Every object visible on the screen has a magnification that depends upon the sequence of portals it is being seen through. As the magnification of an object changes, the user generally finds it useful to see different types of information about that object. For example, when a text document is small on the screen the user may only want to see its title. As the object is magnified, this may be augmented by a short summary or outline. At some point the entire text is revealed. We call this **semantic zooming**.

Semantic zooming works using the *expose event*, which says that a particular portion of the Pad Surface will be rendered at a particular magnification. When an object receives this event it generates the display items needed to give an appropriate appearance at that magnification.

Objects can also manage **portal filters** - portals that show non-literal views of cooperating objects. For example, a portal may show all objects that contain tabular data as a bar chart, but display other objects as would any other portal. This would enable an application to embed a bar chart within a document by placing in it a portal filter that looks onto an object that contains tabular data. Another application can then allow text or spreadsheet style editing of the tabular data itself by some user. These edits will be seen as changes in the bar chart by any user who is looking at the document.

The effect is that the bar chart filter portal will "see" any tabular data as a bar chart, but will see other objects in the usual way. Portal filters work by intercepting the expose event for objects which it knows how to render. It then asks the object or objects for any information it needs to create the display items to render them.

Another interesting portal filter would be a control modifier. Imagine for example that a paint program has several types of brush. Normally one would click on an image of a particular brush to select it. When seen through a control modifier portal filter, each brush image would appear as a panel of parameter controls with which the user can change that brush's internal state (width, spattering law, etc). The *same* portal filter could be used to modify the controls of any application on Pad that recognizes its message conventions.

1.4 Pad vs. the Window/Icon Paradigm

An important distinction between the Pad universe and the universe of other window systems is that in Pad every interaction object possesses a definite physical location. In this sense Pad is a two dimensional virtual reality. Yet a user's changing view can allow objects to *appear* larger or smaller.

This paradigm allows for the sort of peripheral activity found in real physical working environments. Each object on a user's screen commands a degree of attention commensurate with how big the object appears to that user. This allows each object to vary the amount of detail it presents to each user. Different users can share and view multiple applications while assigning to each one a desired degree of interaction. Documents can

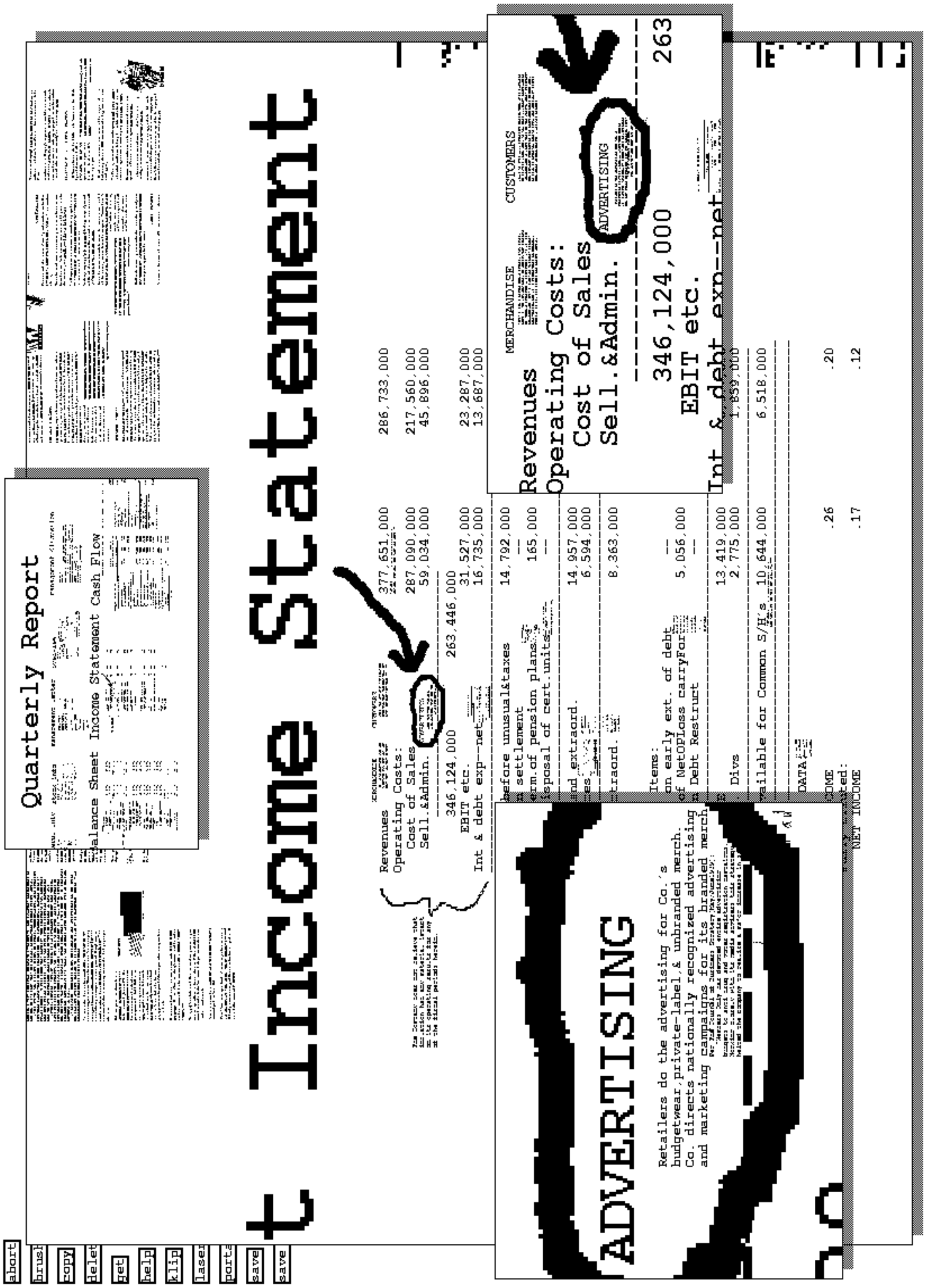

Figure 1: Quarterly report. Portals are views onto other parts of the Pad surface.

be visually nested and zoomed as they move back and forth between primary and secondary working attention. Things can be *peripherally* accessible.

For example, on the Macintosh desktop a user double clicks on a folder icon to see the contents of a directory in a window. But to see the contents of any folder within that folder, the user must double click to create a separate window.

In comparison, a user of Pad generally views a directory through a portal. The contents of any subdirectories are visible, in miniature, through sub-portals. This allows the user a peripheral awareness of a subdirectory's contents, without the user having to perform any explicit action. In this sense, Pad is better suited to non-command user interfaces[16].

1.5 Prior and related work

A number of researchers developed ways to visually structure interactive information that offer an alternative to windows/icons. One of the first such systems was the Spatial Data Management System [4] at MIT, which presented an information landscape on two screens: one screen for a panoramic overview and another (application) screen providing a closer view. The user could either pan locally around on the application screen or else could go directly to an area by pointing on the panoramic view.

On the other hand, Hypertext systems[15][10] allow the user to jump from one place to another in a conceptual information space. A notable problem with the current state of hypertext systems is the difficulty of knowing one's location in this space; unless the application is designed very carefully the user can easily get lost.

In other related work, many desktop publishing systems provide tiny "thumbnail sketches" of images that are stored on disk. To open an image file the user simply points to these miniature images instead of specifying a file name.

A unique approach to providing peripheral information has been developed by George Furnas at Bellcore Applied Research. His **Fisheye** user interface[8] shows information of current interest in great detail, while showing a progressively less detailed view of surrounding information.

Also, some of the components of fast image zooming have existed for a while. Williams[25] has used a pyramid of images for texture filtering, and Burt[2] for image processing, both based on the prior work of Tanimoto[22]. The **Bad Windows** interface[19] allows drawings to be accessed at multiple levels of detail.

Three dimensional interactive virtual offices that allow a user to change viewpoint are being developed by Mackinlay et. al. as well as Feiner [12][6]. Changes of scale have long been used in computer graphics for both entertainment and for scientific visualization.[3] One notable early example was the molecular simulation work of Nelson Max[13].

At Xerox PARC there has been a large body of interesting work on enabling groups to remotely share a common drawing surface for collaborative work.[11][14][21] This is part of their larger ongoing research effort in shared "Media Spaces"[1]. Similarly, the **Rendezvous** system at Bellcore is a general metasystem for building shared "conversational" interfaces for teleconferencing situations[9], as is the work of Smith et. al.[20]

2 An Example Application

The multiscale daily/monthly calendar is a study of "semantic zooming". Figures 2 through 4 show what the calendar looks like at various successive magnifications. At any level, the user can type or draw on the calendar. As the user zooms away from the scale at which the annotations were drawn they become first translucent, then invisible. In this way, a user can overlay many levels of annotation on a calendar without confusion.

The major problem with an application of this type is that it can involve a large number of display items, since the spatial density of display items on the Pad grows geometrically as the user zooms into the calendar. Yet at any one time only a fairly small number of display items is visible, since as the user zooms in the screen occupies an ever smaller absolute area on the Pad.

We address this problem by designing the calendar object as an expandable semantic tree, and identifying display items with different nodes of this tree. Each time the calendar is displayed this semantic tree is traversed. As each node is reached, display items are generated as needed. Individual display items are ephemeral – if an item is off the screen for a while it is quietly removed by the calendar object. In this way the total number of display items always remains manageably small.

This general notion of a geographic database that will expand and self-prune as the user roams around the Pad has now been encapsulated in a Scheme library called an "ephemeral database manager". We plan to apply this library to other Pad applications that have an inherently tree structured semantics.

3 System Structure

In this section we introduce the abstract data types needed to implement Pad. First we will describe the

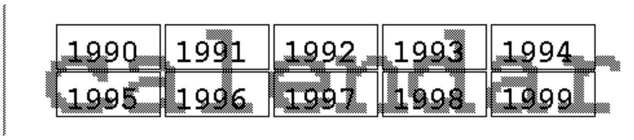

Figure 2: As you approach the calendar object the large scale display items fade out and disappear.

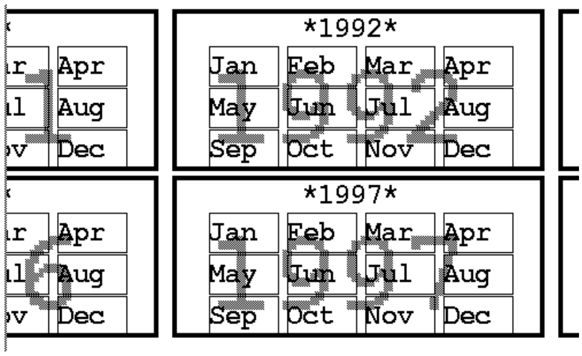

Figure 3: The calendar object generates smaller scale display items only for the area visible on the user's screen. Display items that are off the screen may be garbage collected and destroyed.

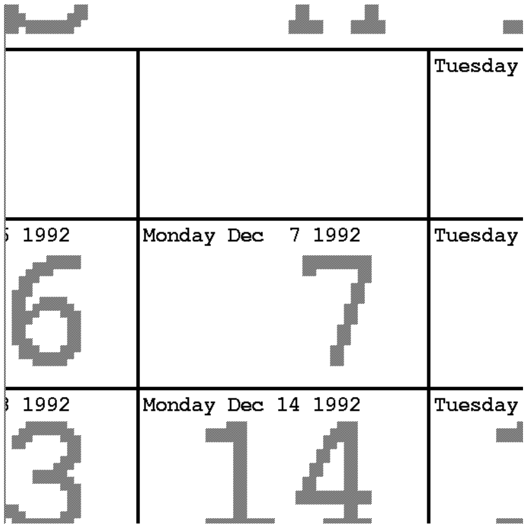

Figure 4: The user's annotations are created in ink that also fades out at greater magnifications.

concepts necessary for display, then those needed to support interaction.

3.1 Addresses and Regions

A Pad **address** $A = (x, y, z)$ has both a location and a scale, and defines the linear transformation $T_A : (u, v) \to (x + u2^z, y + v2^z)$. Here z represents the log_2 of scale.

A Pad **region** $R = [A, w, h]$ is a rectangle defined by an address together with a raster width and height (w, h). A region covers the portion of the Pad surface from $T_A(0, 0)$ to $T_A(w, h)$, or from (x, y) to $(x+w2^z, y+h2^z)$.

3.2 Display Items

The lowest level entities in the Pad universe are the *display items*, which come in two basic types: **graphic** and *portal*. Display items are the only entities actually visible on the user's screen. A *graphic* consists of a raster image I and an address A. Every display item is said to have a region $[A, I_w, I_h]$, which is the portion of the pad surface which it occupies.

A *portal* is a graphic that has an additional address, called its *look-on* L. Using its raster image I as a mask, a portal have as its "look-on" the region $[L, I_w, I_h]$ on the Pad surface. The portion of the Pad surface which the look-on covers and which is not masked by the portal's graphic is visible at the location of the portal's region. This raster masking enables a portal to give a shaped view onto the Pad surface. Thus, a portal can be square, round, or even shaped like some well known corporate logo.

We refer to a display item's A_z as its "scale". In general, a display item becomes visible on the screen only after being viewed through a succession of portals, each of which may transform it. We refer to a display item's apparent z, as it is seen on the screen, as its "magnification".

The image on the user's screen is created from a set of display items. There is one portal associated with the user's screen called the "root portal"; the display process consists of rendering the root portal. This means rendering the region of the Pad surface which the root portal looks onto. Those display items that overlap the root portal's look-on are rendered. This procedure is then applied recursively to render any display item which is itself a portal.

As the display process recurses through each portal, the transformation $T(A)T^{-1}(L)$ is applied, where A is that portal's address and L is that portal's look-on. This recursion can be expanded to compute the location of any display item on the screen. Suppose item i is viewed through successively nested portals $p_1 \ldots p_n$. Then to determine where (and at what magnification) to display i on the screen, we apply the transformation:

$$T^{-1}(L_{root})T(A_{p_1})T^{-1}(L_{p_1}) \ldots T(A_{p_n})T^{-1}(L_{p_n})T(A_i)$$

Incrementing the z component of a display item's address will increase its magnification. Incrementing the z component of a portal's look-on will double the size of its looked on region – and will therefore decrease the magnification of every item seen through it. (Think of it as increasing the viewer's altitude.)

There are several other properties of primitive display items which are important to note:

Visibility Range: Each graphic object can have a range of magnification outside of which it is invisible. This is important since most display items are only useful within a certain range of magnification.

Transparency Range: Similarly, each graphic can have a range of magnification outside of which the graphic is transparent. This allows objects to fade away gracefully as they are magnified up or down. Transparency is achieved by masking with a patterned pixel mask at screen resolution.

Private Display Items: Display items may be attached to a portal, in which case they are only visible when viewed through that portal and their addresses are relative to that of the portal. This creates a hierarchy of display items and is used to implement the filters described below.

3.3 Pad Objects

Graphics and Portals suffice to make an interesting multi-scale drawing program. However to use Pad as a system for building general user interfaces requires a higher level structure called a *Pad Object* to interpret events and control these display items so they behave as a single application. In Pad an object consists of a region together with a package of code and data which respond to event messages. An object's behavior is specified by the application developer. In order to make itself seen, each object manages a collection of display items, creating, modifying, and deleting them.

Pad Objects receive events from the user's mouse and keyboard, plus timer events, channel events (events representing other types of input, e.g. the output of a process), and expose events which inform the object that some portion of itself will become visible on someone's screen. Events which would normally have an x-y location have instead an *address*, and this address is transformed if the event passes through a portal before being received by an object which is interested in it. Similarly, an expose event covers a *region* rather than just a rectangle, and this region is also transformed by portals so that each object can be informed which portion of its region will be rendered and at what magnification.

Objects are maintained in an order, just as display items have a drawing order, so that if two or more objects are at the mouse address the mouse events are sent to the one in front. The object may use this event for its own purposes, or it may pass the event on to the objects behind it, or it may transform the event's address and pass it on to some other part of the Pad. Events thus passed may go unused by the objects below, in which case the original object may then use the event for its own purposes.

3.4 Display

Display is complicated by the fact that objects may be continually creating and destroying display items. Before we can create the display we first need to give each object an opportunity to know at what magnification it will be called upon to appear, since this will probably influence what display items it chooses to show.

Therefore display is a two phase process. In the first phase, each object gathers all the necessary information about what portions of it will appear on the screen and at what magnifications. During this first phase display items may be spawned. In the second phase the screen image is actually drawn.

During phase one each portal is displayed by having the Pad object that controls it communicate with all objects that intersect the portal's look-on region. This process begins with a special root object, which controls the user's root portal. For a portal controlled by an object O_1 the procedure is as follows:

- O_1 sends an expose event for the portal's look-on region. This event will be received by all objects whose regions intersect the portal's look-on region.
- for each object O_2 that responds:
  - O_1 tells O_2 to produce display items for itself with the proper magnification and clip. If O_2 controls any portals, the procedure is invoked for them recursively.
  - any display items that O_1 receives back, it attaches to the portal.

This process continues recursively until all items large enough to see on the screen are accounted for.

In the second phase, each portal is painted from its accumulated list of display items. This process starts with the root portal, and continues on through all portals seen by the root portal, and then recursively through those portals. Note that if two portals on the screen have overlapping look-on regions, their lists may have display items in common.

3.5 Interacting Objects and Portals

Semantic zooming is implemented by having the object's display method depend upon its magnification. The object is always told its magnification during display phase one.

Portal filters are implemented as follows. Consider the case of the bar chart filter portal described earlier. Suppose this portal filter is managed by object O_1. During phase one of the portal display procedure, O_1 sends an expose event for this portal, and receives a number of acknowledgments. Suppose O_1 has just received such an acknowledgment from object O_2. O_1 queries O_2 to find out whether O_2 is a tabular object. If yes, then O_1 gets the tabular data from O_2, builds its *own* display items for the bar chart, and attaches these to the portal. If no, then O_1 asks O_2 to produce a list of display items as usual. The effect is that the filter portal will "see" any tabular data as a bar chart, but will see other objects in the usual way.

4 Implementation Details

The Pad system is written in three layers, a real-time display layer written in C++, a Scheme interpreter providing an interface to the C++ layer, and a collection of Scheme code implementing the Pad application interface. It currently runs under X Windows and MS-DOS. The X Windows version has been compiled and run on SunOS, AIX and Linux. The source code of the most recent released version is available via anonymous FTP from cs.nyu.edu in the directory pub/local/perlin.

4.1 Rendering Display Items

It is absolutely essential to our system that arbitrarily scaled bitmaps can be displayed in real time. Without an algorithm to achieve this, our desktop model would either require special purpose hardware, or else would lose real-time response. Either scenario would limit the model's general usefulness on typical currently available graphical workstations. The method we use to render the raster image of a graphic item depends upon the item's magnification. The following decisions are based on our trial and error experiences; they reflect our best results in "tuning" this process.

We use four different techniques for drawing the raster image of a graphic, depending on the range of magnification m.

- $m > 16$. At the largest magnifications it is quickest to simply draw individual filled squares for each pixel.
- $1 > m \geq 16$. At moderate magnifications we use look up tables indexed by the byte pattern, amount of magnification, and bits of shift to properly position the result within the destination word. Different tables are used depending on the depth of the image.
- $m = 1$. With no magnification we only need to worry about the amount of shift necessary to position the result.
- $\frac{1}{1024} \leq m < 1$. To demagnify images we index into a pre-computed pyramid of images.[25] This precomputation is done at the time a graphic is created; it creates about a 3/2 speed penalty to that process. Since graphic items are generally reused over many screen refreshes, this penalty is not usually a problem in practice.
- $m < \frac{1}{1024}$. Beyond some amount of demagnification the bitmap is not visible and need not be drawn at all.

These techniques yield a display time for each object approximately proportional to the size of the entire screen image. In practice this tends to keep refresh time dependent only upon screen resolution, not upon image complexity.

4.2 Address Space Limits

Addresses are implemented using floating point arithmetic, so we cannot claim an "infinite" address space for our current system. A true unbounded address space could be achieved by using extended integer arithmetic. Even in its current form, the space provided is astronomical. Suppose our numbers have a 48 bit mantissa and we have a 2^{12} by 2^{12} screen. To position an object on the screen uses 12 of those 48 bits, leaving a minimum of 36 bits of precision to position our look-on anywhere within the square $-1 \leq x, y \leq 1$. This means, for example, that you could lay out 2^{36} by 2^{36} pages of text in that area.

5 Ongoing and Future Work

5.1 Shared Object Space

Perhaps our most important goal is to create a truely distributed Pad system, where Pad objects can exist on remote machines and can migrate from machine to machine. When Pad objects are distributed over many computers the problem of updating the display of a region on one's screen becomes a combined distributed database and computational geometry problem. This is the subject of ongoing research,[7] and is beyond the scope of this paper. For in-depth discussions of the implementation problems we refer the readers to Preparata & Shamos[18] for an overview of computational geometry and to Edelsbrunner[5] for an optimal data structure for rendering.

5.2 Continuous Zoom

Early prototypes have used discreet zoom levels to achieve high performance. We have also implemented a continuous zoom algorithm (based on Bresenham's midpoint line drawing algorithm) that allows continuous scaling of raster images at approximately half the speed of discreet zooming on unenhanced bitmapped workstations. The algorithm uses table lookups to greatly speed up the calculation.

5.3 Hierarchical Text Editor

A number of generalizations of familiar applications to the hierarchical domain suggest themselves. A multiscale text editor is a generalization of a traditional text editor, with the added capabilities that text can appear at many different sizes, with recursively inserted text. Therefore the screen structure is no longer a two dimensional array - it is more like a set of nested boxes. This allows a more direct look-and-feel for hypertext - footnotes and references can be embedded in their entirety at the point of reference. Successive zooming by the user gradually expands the contents seen of the work referenced. Text is structured as hypertext – a text string may contain embedded links to other text strings. The structure of the document can be an arbitrary directed graph. Visually, text that is linked to appears to be at the location of the link, only smaller. Contents of a hyperlink can be accessed without a disruptive sudden change in the view of the text that references it.

Text can also be made semantically zoomable: When text is visibly small it appears only as a title. As the user zooms in, this expands to include an abstract. Further zooming reveals first an outline with short text descriptions, then finally the full text.

There are several options for where exactly to visually place linked-to text. The text can appear in miniature either beneath the lines of parent text or, alternatively, superimposed on the parent text. The latter option requires zoom-dependent translucency. As the user zooms in, text seen through hyperlinks "fades up" and the visually larger text that references it simultaneously "fades out".

Text can be visible simultaneously in any number of portals. Each view must maintain a certain amount of state information. For example, there needs to be a cursor for each view. This means that if the mouse is over a particular portal, and the user types, the insertion point is at the cursor of that view. Since portals can contain ownership attributes, they can be used to restrict access to parts of a document. Text visibility through any particular portal depends upon the text's ownership - public (shared by many users) or private (seen by only one user). Public text can contain links to private text. In general, the visibility attributes of text can vary, depending upon whether the text is being viewed by its owner or by someone else.

5.4 An Infinitely Scalable Painting Program

We have, together with Luis Velho, begun applying multiscale principles to an infinitely detailable painting program[17]. Organizing an infinite multiscale canvas is straightforward, requiring only a Quad-tree. Unfortunately, simulating the application of a paint brush requires a compositing operation - an *alpha* blending of the underlying image with the brush image.

Since this operation is non-commutative, it is easy to run into problems. For example, Let's say the user zooms way in to paint a scene at a fine scale, then pulls out to paint an atmospheric wash at a coarse scale, and finally zooms back in to touch up fine scale details. How should the system implement this? A straightforward approach, used by Williams[26], is to immediately apply the coarse scale operations to the finer level pixels. But this is computationally prohibitive for highly scaled scenes, since the number of fine scale pixels affected grows exponentially with the difference between coarse and fine scale.

Clearly a pyramid of some kind is called for. But because of non-commutativity, successive operations at different levels cannot be separated into a traditional Laplacian or similar multilevel pyramid (as they could be in, say, a strictly additive system). Our solution is to use B-spline wavelets. We break the brush image into its component wavelet basis, and apply independently at each level of a wavelet basis pyramid. Then the B-spline wavelet reconstruction will produce the correct result. We have implemented this to a one-dimensional canvas, and are now working on a two or more dimensional version.

5.5 Multiple Narrative Paths

Pad is a good way to store documents with hierarchy and multiple narrative pathways. Side discussions in a textbook can be embedded *in situ*. This allows for some interesting possibilities. For example, a novel may be written with bifurcations, allowing its reader to explore

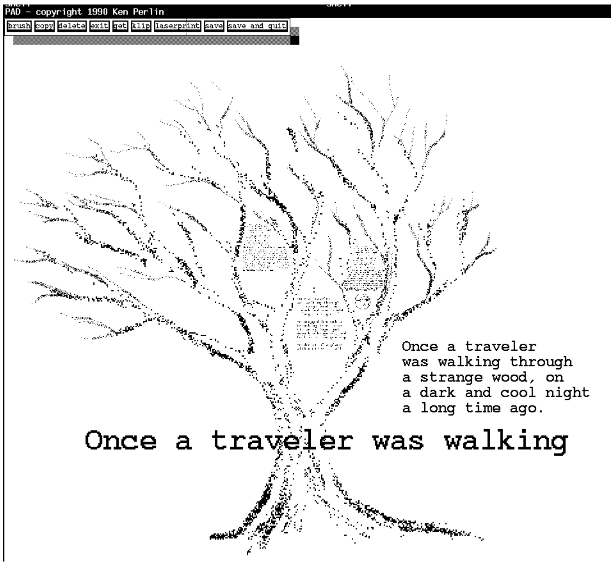

Figure 5: Overview of branching tree story. The story begins with a single sentence. The branches of the tree represent story paths - as the reader zooms into different branches, different stories unfold.

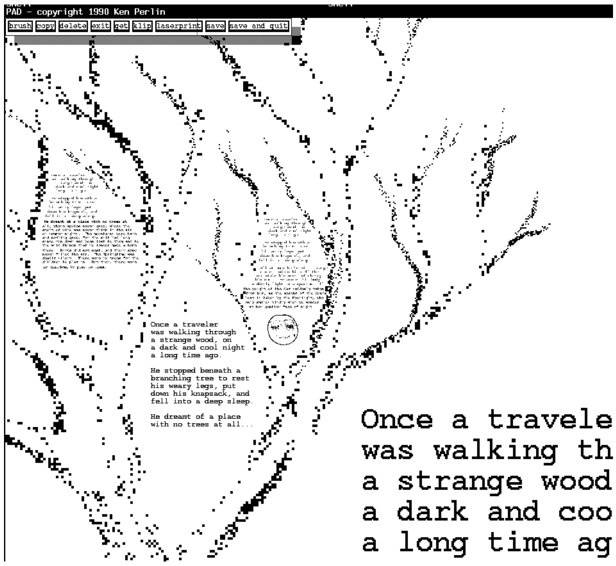

Figure 6: One level of zoom into branching tree story. At this scale the narrative contains one or two paragraphs of detail.

many interleaving stories - a sort of visual *Alexandria Quartet*. For example, we have been creating a user browseable novel literally shaped into a tree, as seen in figures 5, 6, 7.

5.6 Cooperative Pad Applications

With the onset of high bandwidth consumer information services, Pad provides a viable look-and-feel for information browsing. As the customer zooms in to an information service, the semantic zoom level (and hence the information content) increases. Zoomed-down browsing can be made freely available, and the customer can be billed at successively higher rates for more specific data.

For example, the title and a brief synopsis of a video may be accessible at low zoom levels. Higher zoom levels actually play the movie. At the browsing level, the customer might see geographically arranged clusters of films that may be of related interest (e.g. films by a particular director).

Similarly, our **Pad Map** project will provide a substantial user community with access to a shared map of Manhattan, annotated with information about cultural events. The users will be able to add their own annotations, such as restaurant or movie reviews, or just graffiti. As part of the Pad system, annotations could be at any scale, and contain links to other annotations: though it is desirable to keep all the reviews of a given film together, portals could make them visible at each theatre which is showing that film. The project will explore the mechanisms necessary manage user contributions without any one user monopolizing or degrading the system for others.

Our **Shared Spreadsheet** project re-casts the spreadsheet application in a more hierarchical and sharable form. For example, hierarchy can be imposed by placing spreadsheet A in a cell of spreadsheet B, and designating a particular cell of A to be the value that appears in B's cell when the magnification of A is low. The value of sharing such a spreadsheet among users comes from immediate access to the latest data, and the elimination of the need to merge copies of the spreadsheet which have been updated independently, etc.

Eventually, as display and communication technology improves, pieces of display surface scattered around a work environment will become more common – on walls, desks, electronic PostIt^{tm} notes[24]. Pad is well suited to such a distributed environment, since it places the user at a floating location in an information geography. The Windows/Icon/Menu/Pointer model is less well suited to this, since it is motivated by the desire to create a "desktop" metaphor on a single display screen.

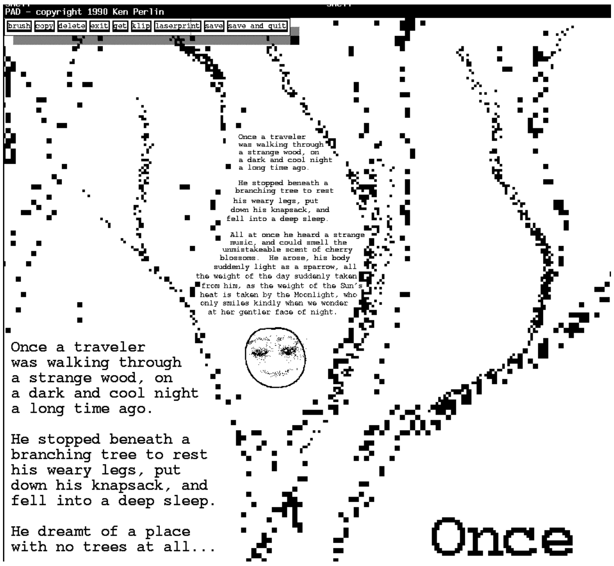

Figure 7: Two levels of zoom into branching tree story. Here we can see the story beginning to take a definite shape - in one possible narrative path.

6 Conclusions

We have described a new kind of graphical space that has a number of advantages over traditional window systems. Its key advantage is that it allows a user or a group of users to share and view multiple applications in a manner that assigns them various levels of importance, with easy visual nesting and zooming of documents as they move from peripheral to primary working attention.

As compared to standard current window models, this system makes it easier for the user to exploit visual memory of places to organize informationally large workspaces.

We believe that this approach enriches the workstation/window paradigm in a fundamental way.

6.1 Acknowledgments

This research was funded by a grant from the NYNEX Corporation and by NSF grant number IRI-9015445. We would like to thank Nathan Felde at NYNEX for the initial discussions leading to this work, and Jack Schwartz, Lorie Loeb, Raj Raichoudhury, Allison Druin, and Gene Miller, all of whom contributed valuable ideas and time, as well as the Apple corporation for their generous equipment donation. Particular credit goes to Matthew Fuchs, who is developing the Distributed Pad/Scheme system DREME.

References

[1] Bly, Sara et. al. (1993) *Media Spaces: Bringing People Together in a Video, Audio, and Computing Environment.*, CACM, Vol. 36, No. 1., pp. 28-47.

[2] Burt, Peter. (1983) *A multiresolution spline with applications to image mosaics,* ACM Transactions on Graphics, Vol. 2, No. 4, pp. 217-236.

[3] Clark, James H. (1976) *Hierarichical geometric models for visible surface algorithms.* ACM Communications, Vol. 19, No. 10, pp. 547-554.

[4] Donelson, William C. (1978) *Spatial Management of Information,* ACM SIGGRAPH 1978 Conference Proceedings.

[5] Edelsbrunner, H. (1983) *A new approach to rectangle intersections, Part II,* Int'l Journal of Computational Mathematics, No. 13, pp. 221-229.

[6] Feiner, S., Beshers, C. (1990) *Worlds within worlds: Metaphors for exploring n-dimensional virtual worlds.* Proc. UIST '90 (ACM Symp. on User Interface Software and Technology), Snowbird, UT, Oct. 3-5, 1990, pp. 76-83.

[7] Fuchs, Matthew. unpublished Ph.D. dissertation in progress.

[8] Furnas, George. (1989) *Generalized Fisheye Views, Human Factors & Computer Systems,* CHI 89 Conference proceedings, pp. 16-23.

[9] Hill, Ralph, et. al. (1993) *The Rendezvous Language and Architecture,* CACM Vol. 36, No. 1., pp. 62-67.

[10] *Hypertext on Hypertext,* Macintosh Version: Disk #1 and #2. ACM Press, New York, 1988.

[11] Lu, I., et. al. (1991) *Idea management in a shared drawing tool.* Proceedings of the Second European Conference on Computer-Supported Cooperative Work-ECSCW '91, Amsterdam, Holland.

[12] Mackinlay, J., et. al. (1990) *Rapid Controlled Movement Through a Virtual 3D Workspace.* ACM SIGGRAPH 1990 Conference Proceedings.

[13] Max, Nelson. (1975) ACM SIGGRAPH 1975 Film show.

[14] Minneman, S., & Bly, S. A. (1991) *Managing a trois: A study of a multi-user drawing tool in distributed design work* Proceedings of the CHI'91 Conference on Human Factors in Computer Systems., New Orleans, La.

[15] Nelson, Ted. (1981) *Literary Machines.* Swarthmore, PA.

[16] Nielsen, Jakob. (1993) *Non-command User Interfaces,* CACM, Vol. 36 No. 4, pp. 83-99.

[17] Perlin, Ken, & Velho, Luis. *A Wavelet Representation for Unbounded Resolution Painting*, NYU Technical Report.

[18] Preparata, Franco P., & Shamos, Michael Ian. (1989) *Computational Geometry: An Introduction*, Springer Verlag, New York.

[19] Small, David. (1989) Masters Thesis, MIT Media Laboratory.

[20] Smith, Randall B., O'Shea, Tim, O'Malley, Claire, Scanlon, Eileen, & Taylor, Josie. (1989) *Preliminary Experiments with a distributed, multi-media, problem solving environment.* In Proceedings of the First European Conference on Computer Supported Cooperative Work (Gatwick, UK) 1989, pages 19-34.

[21] Tang, J. C., & Minneman, S. L. (1990) *Videodraw: A video interface for collaborative drawing.* Proceedings of the CHI '90 Conference on Human Factors in Computing Systems, Seattle, Wash., 1990.

[22] Tanimoto, S. L., & Pavlidis, T. (1975) *A hierarchical data structure for picture processing.* Computer Graphics and Image Processing, Vol. 4, pp. 104-119.

[23] Tufte, Edward. (1983) *The Visual Display of Quantitative Information,* Graphics Press.

[24] Weiser, M. (1991) *The Computer for the 21st Century*, Sci. Am. 265,3, pp. 94-104.

[25] Williams, Lance. (1982) *Pyramidal Parametrics.* ACM SIGGRAPH 1982 Conference Proceedings.

[26] Williams, Lance. Personal communication.