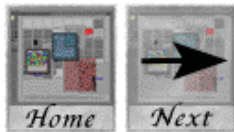


A Brief Tour Through Pad++

Jonathan Meyer, April 1997

- [The Pad++ Metaphor](#)
 - [What exactly is meant by zooming?](#)
- [Why zoom? Three example applications](#)
 - [1. An outline viewer](#)
 - [Interaction with the outline view](#)
 - [Hypermedia presentations using outlines](#)
 - [2. A file browser](#)
 - [Semantic zooming in the file browser](#)
 - [3. An HTML browser](#)
 - [Preserving history in the HTML browser](#)
- [Other Pad++ features](#)
 - [Portals - twists in Pad++ space](#)
 - [Portals vs Windows](#)
 - [Hyperlinks](#)
 - [Sticky objects](#)
 - [Sticky portals as indexes](#)
 - [Portal filtering and 'magic lenses'](#)
 - [Other uses for lenses](#)
- [Conclusion](#)

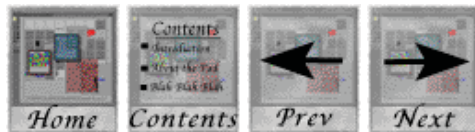


The Pad Metaphor

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 Pad++ Metaphor (continued)

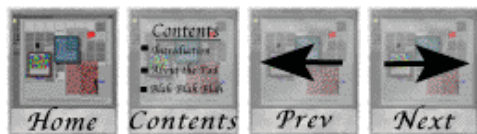
The above scenario is impossible because we can't read or write at microscopic scale. Yet the concept is very natural since it mimics the way we continually manage to find things by giving everything a physical place. A good approximation to the ideal depicted would be to provide ourselves with some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

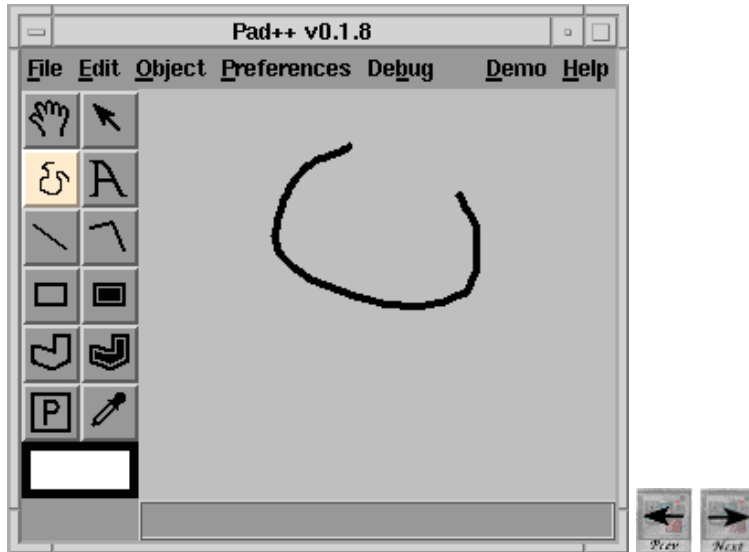
Pad++ gives users this zoomable surface. Before we look in detail at Pad++, though, let's clarify what we mean by zooming.

What exactly is meant by zooming?

To illustrate what we mean by zooming, there follows a series of snapshots of a Pad++ session which show how you might interact with a zoomable drawing application.

Click on the Next button below to read on.



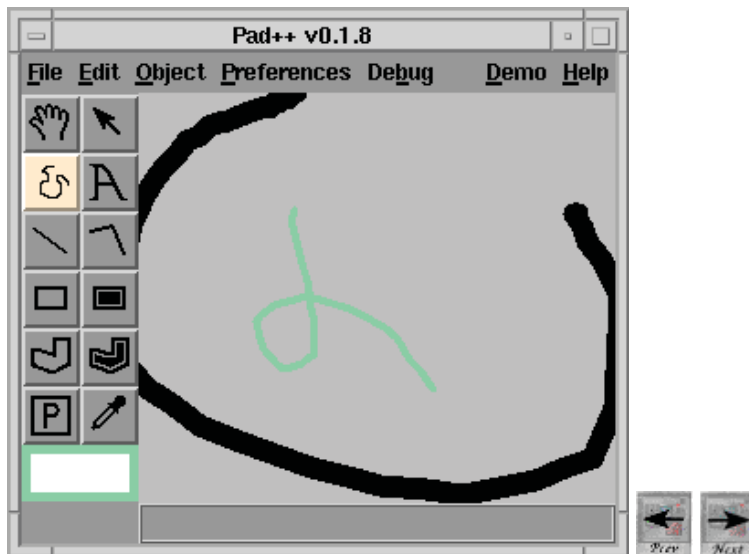


Here we see a simple shape drawn on surface of our Pad++ drawing application.

In Pad++, clicking and holding on the middle mouse button performs an smooth animated zoom in.

Your browser, unfortunately, can't show this! Instead, click on the next button above to single-step to the next image in the sequence...

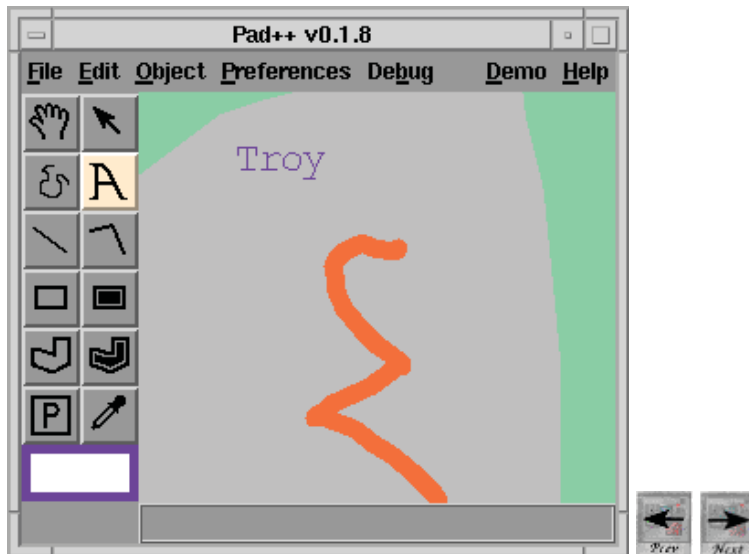
Click on the next button to go on.



After zooming in a little, we have drawn a second, smaller shape.

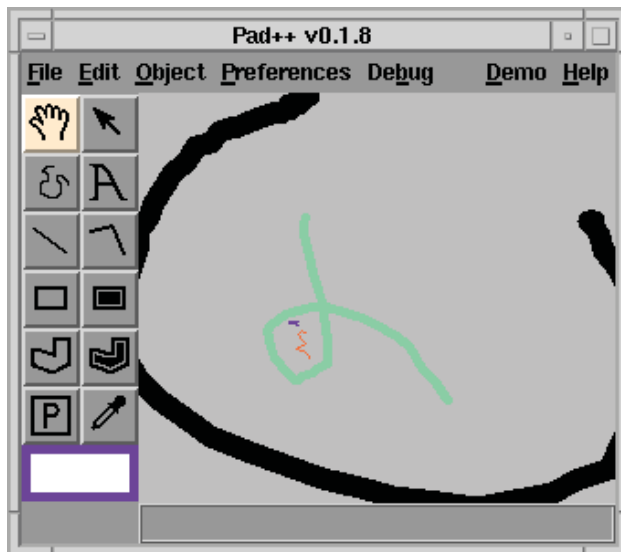
This process can be repeated indefinitely...

Click on the next button to go on.

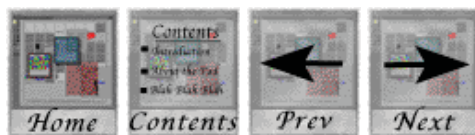


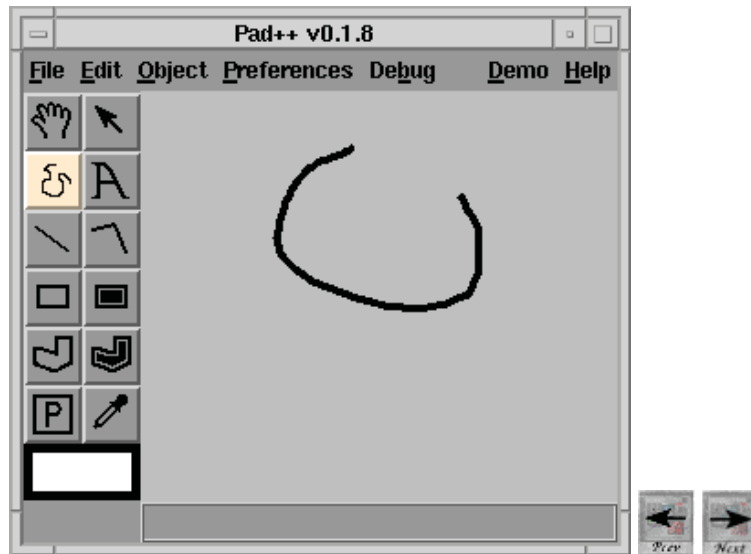
Here we have zoomed in a little more and drawn a third shape and also some text.

Click on the next button to go on.



At any point, you zoom out to see an overview of what's been done.



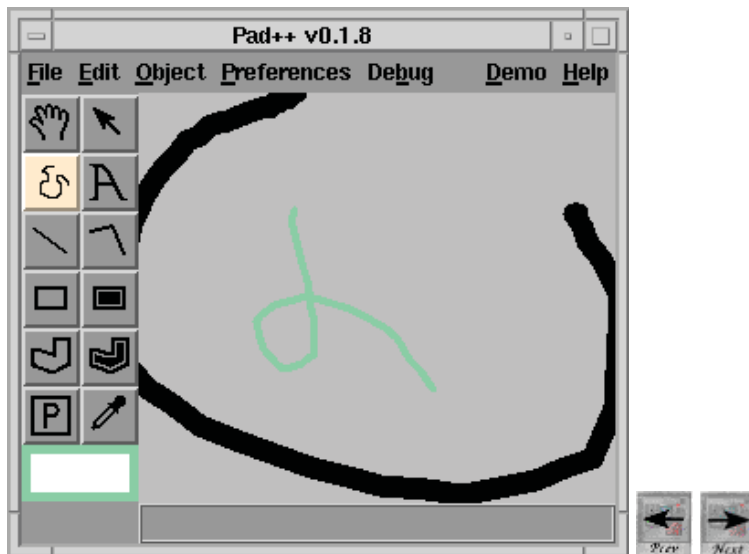


Here we see a simple shape drawn on surface of our Pad++ drawing application.

In Pad++, clicking and holding on the middle mouse button performs an smooth animated zoom in.

Your browser, unfortunately, can't show this! Instead, click on the next button above to single-step to the next image in the sequence...

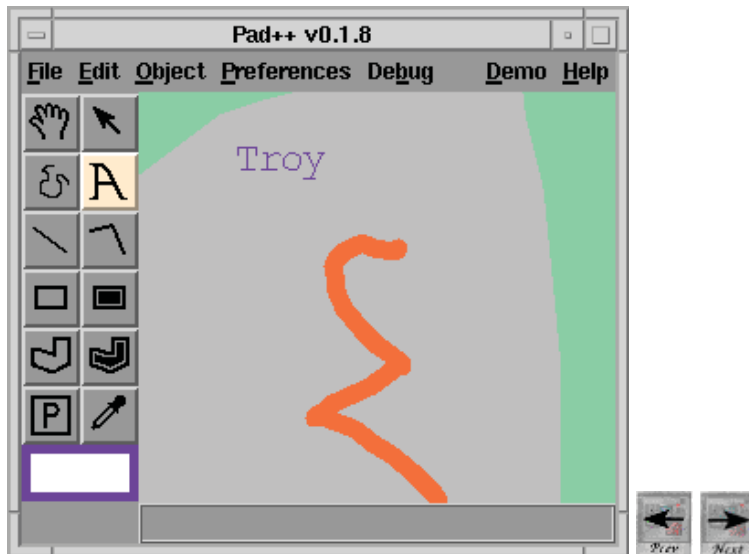
Click on the next button to go on.



After zooming in a little, we have drawn a second, smaller shape.

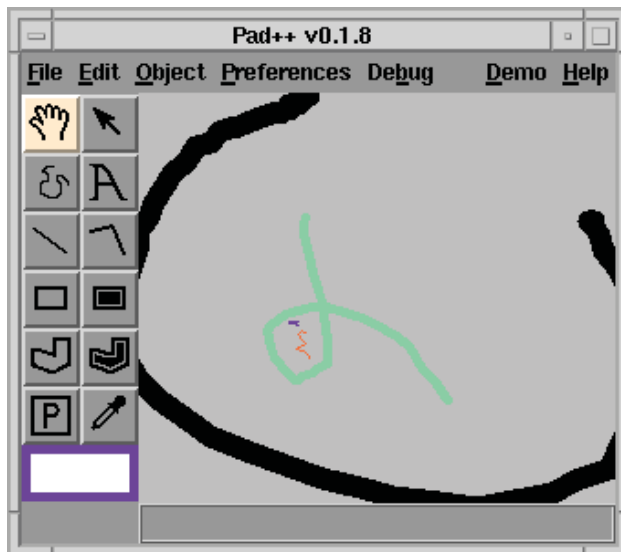
This process can be repeated indefinitely...

Click on the next button to go on.

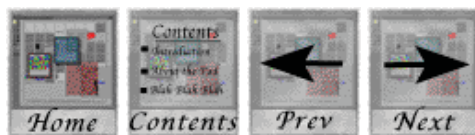


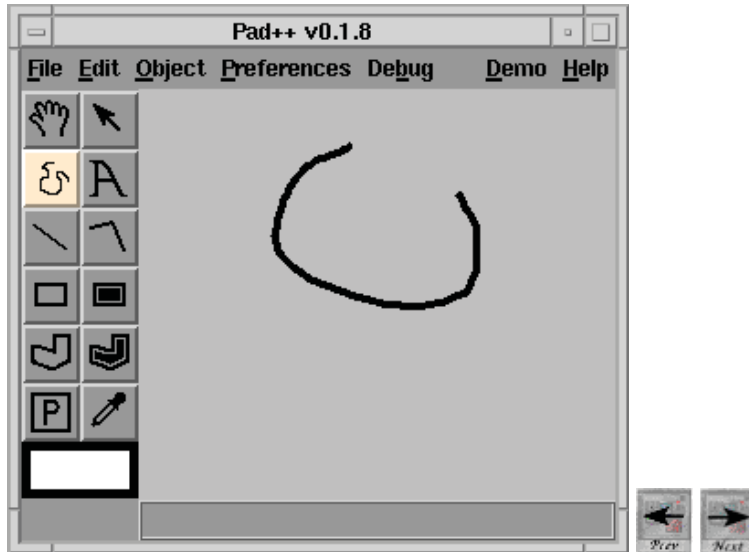
Here we have zoomed in a little more and drawn a third shape and also some text.

Click on the next button to go on.



At any point, you zoom out to see an overview of what's been done.



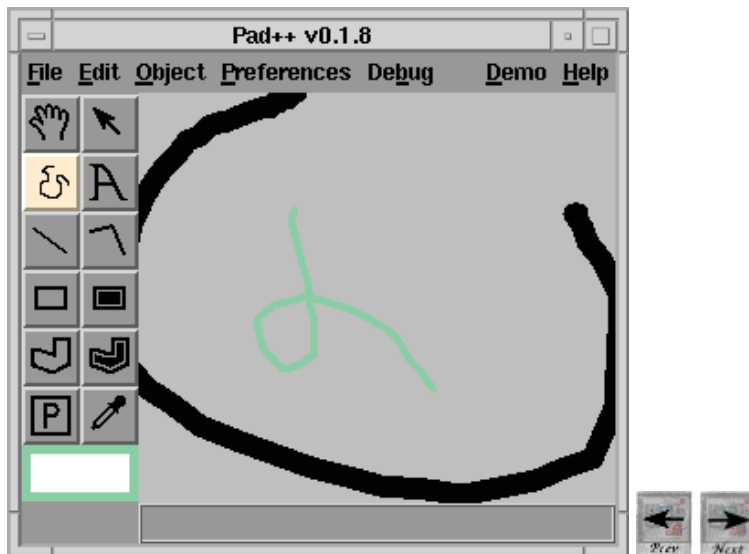


Here we see a simple shape drawn on surface of our Pad++ drawing application.

In Pad++, clicking and holding on the middle mouse button performs an smooth animated zoom in.

Your browser, unfortunately, can't show this! Instead, click on the next button above to single-step to the next image in the sequence...

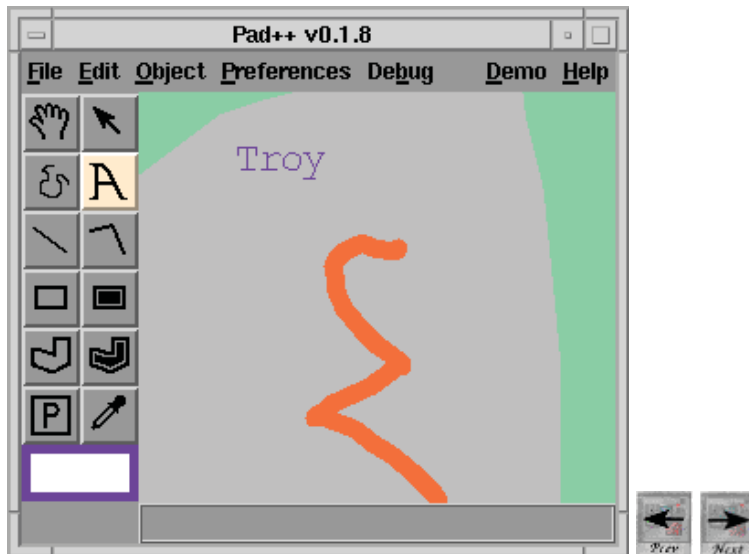
Click on the next button to go on.



After zooming in a little, we have drawn a second, smaller shape.

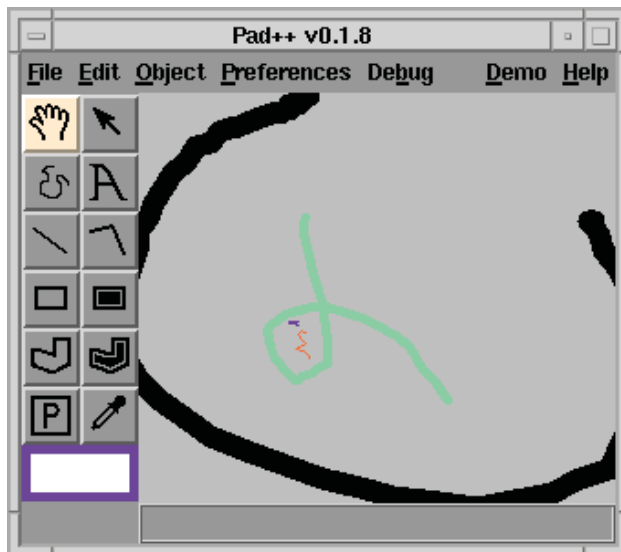
This process can be repeated indefinitely...

Click on the next button to go on.

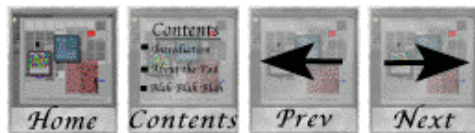


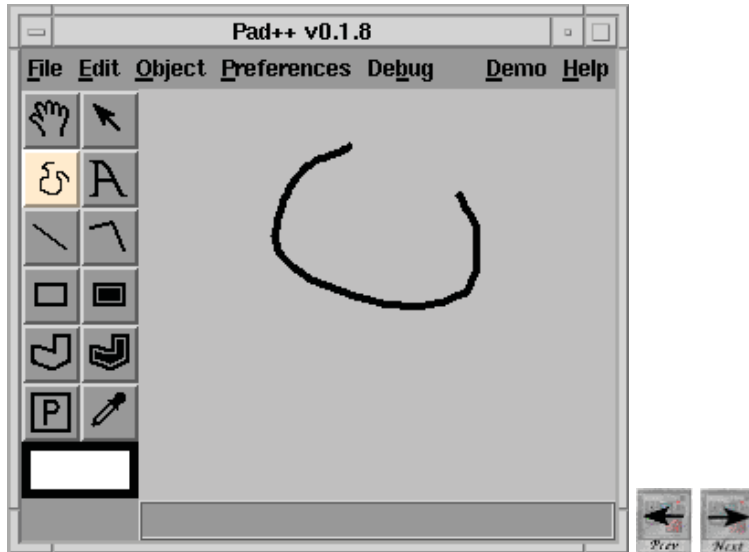
Here we have zoomed in a little more and drawn a third shape and also some text.

Click on the next button to go on.



At any point, you zoom out to see an overview of what's been done.



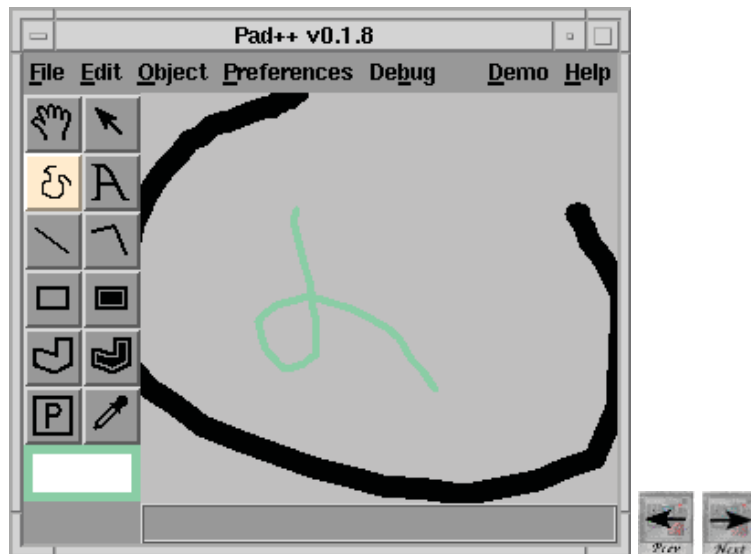


Here we see a simple shape drawn on surface of our Pad++ drawing application.

In Pad++, clicking and holding on the middle mouse button performs an smooth animated zoom in.

Your browser, unfortunately, can't show this! Instead, click on the next button above to single-step to the next image in the sequence...

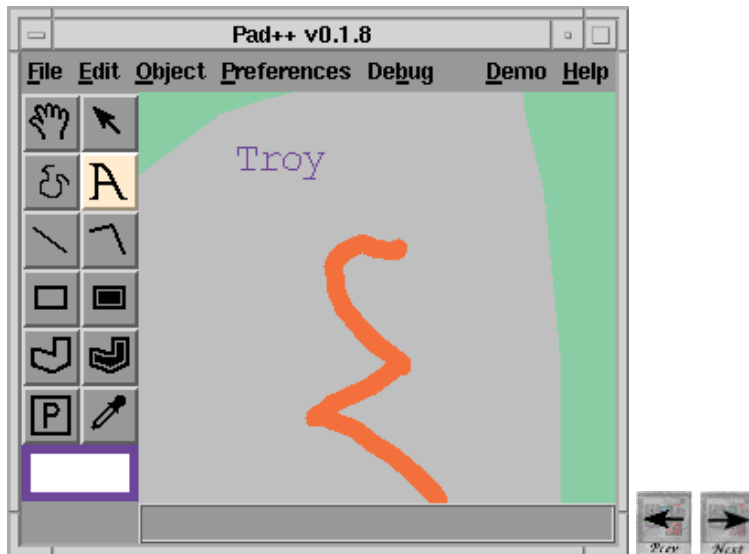
Click on the next button to go on.



After zooming in a little, we have drawn a second, smaller shape.

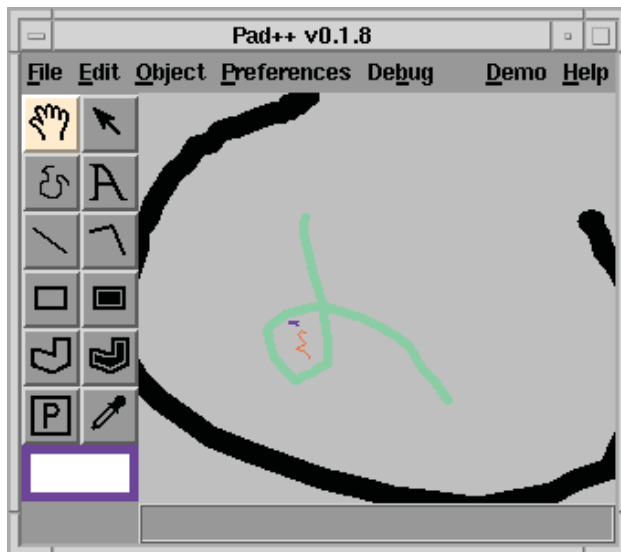
This process can be repeated indefinitely...

Click on the next button to go on.

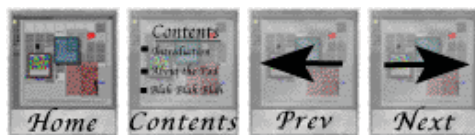


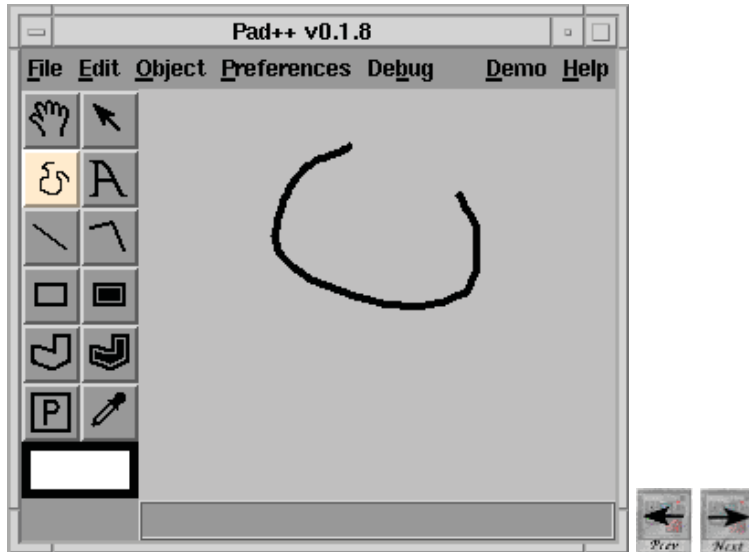
Here we have zoomed in a little more and drawn a third shape and also some text.

Click on the next button to go on.



At any point, you zoom out to see an overview of what's been done.



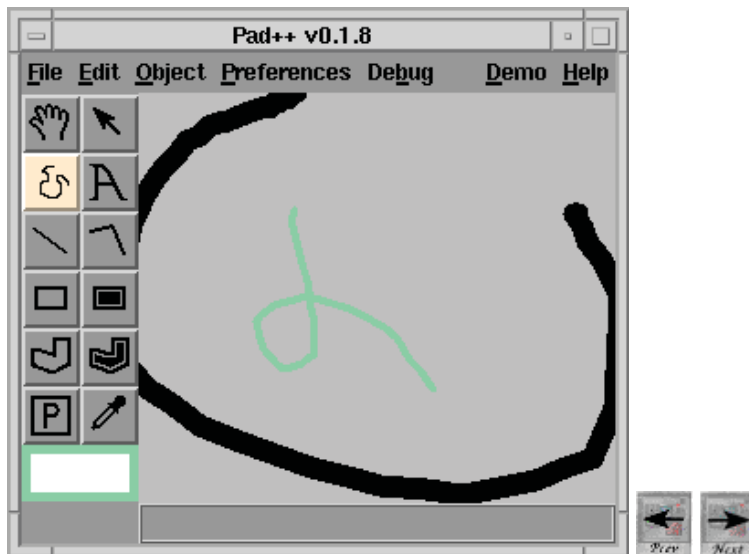


Here we see a simple shape drawn on surface of our Pad++ drawing application.

In Pad++, clicking and holding on the middle mouse button performs an smooth animated zoom in.

Your browser, unfortunately, can't show this! Instead, click on the next button above to single-step to the next image in the sequence...

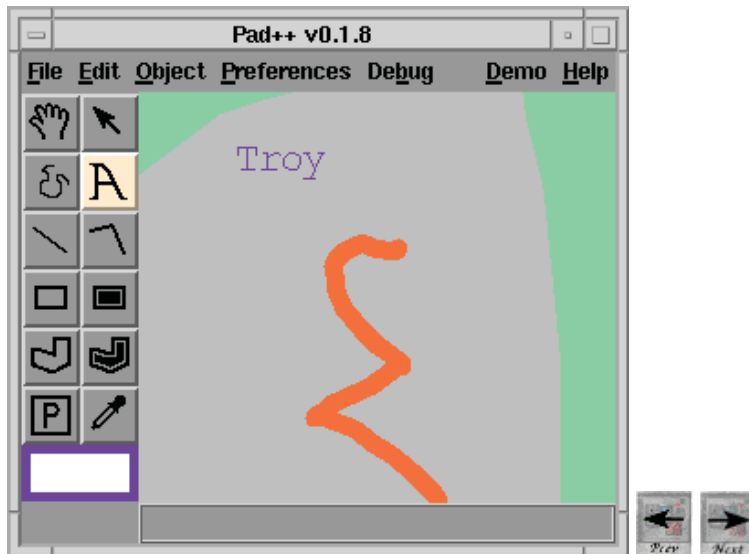
Click on the next button to go on.



After zooming in a little, we have drawn a second, smaller shape.

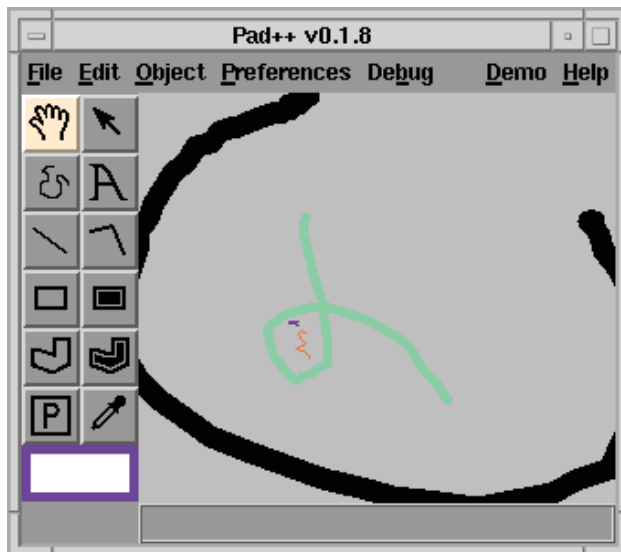
This process can be repeated indefinitely...

Click on the next button to go on.

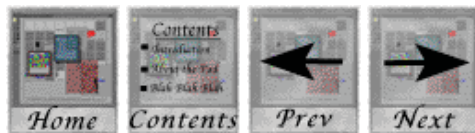


Here we have zoomed in a little more and drawn a third shape and also some text.

Click on the next button to go on.



At any point, you zoom out to see an overview of what's been done.



Why zoom? Three example applications

Once you add zooming to a user interface, new kinds of interactions are possible. The following sections introduce three sample applications that we have developed which use zooming.

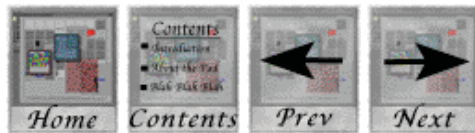
1. An outline viewer

The Pan/Zoom space is a powerful visualisation for any kind of hierarchical data - nested details can be shown at a smaller scale, conveying the relations between objects very naturally.

For example, we have developed an interactive outline viewer which we use for giving presentations. The outline viewer shows the text of the whole presentation - you simply zoom in on the points you want to make.

Main topic headings are shown at a large scale, subheadings are shown at a smaller scale, and individual points are shown even smaller.

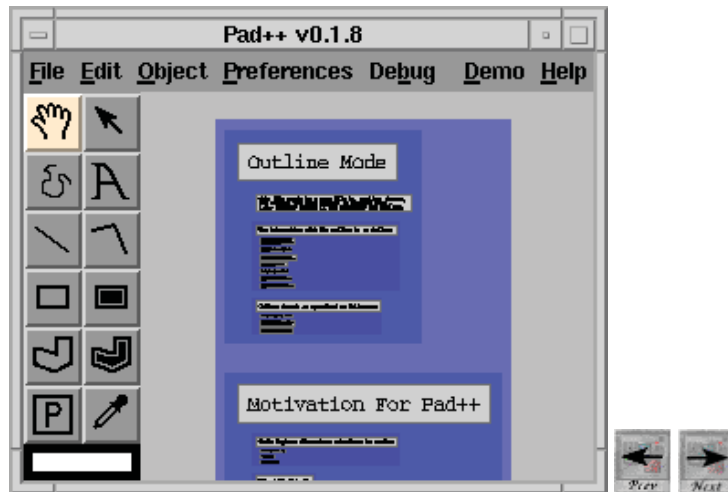
Click on the Next button below to read on.





For example, here we see a distant view of an outline for a presentation about Pad++ itself.

Click on the next button to go on.



When you zoom in, the major headings become legible.

Click on the next button to go on.



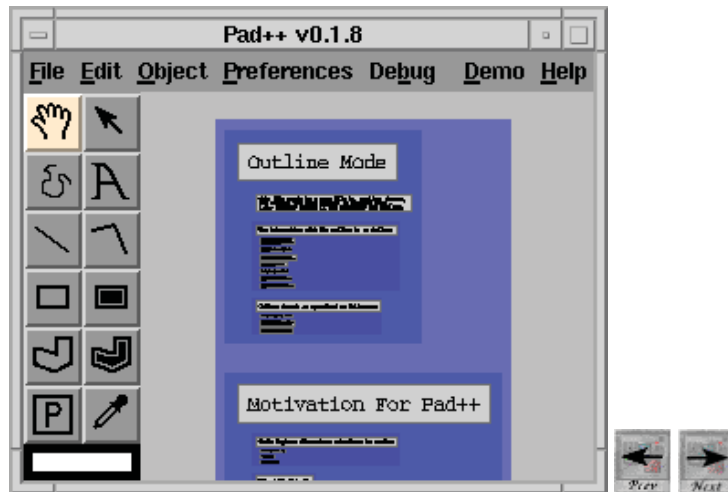
When you click on a piece of text, Pad++ zooms in so that that text fills the screen horizontally and appears near the top of the screen vertically.





For example, here we see a distant view of an outline for a presentation about Pad++ itself.

Click on the next button to go on.



When you zoom in, the major headings become legible.

Click on the next button to go on.



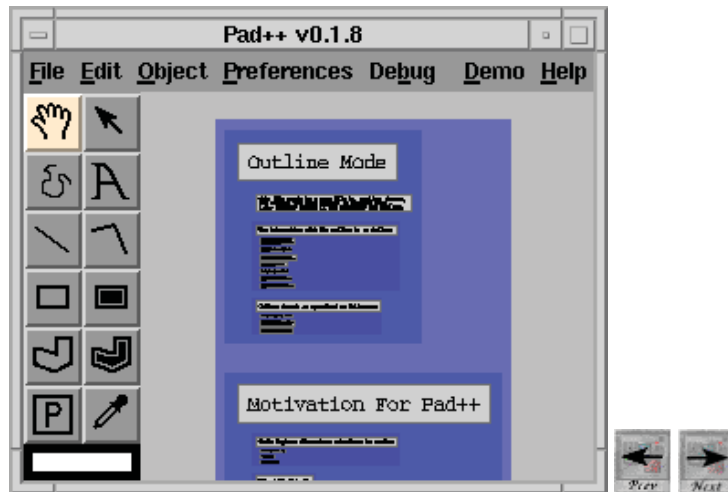
When you click on a piece of text, Pad++ zooms in so that that text fills the screen horizontally and appears near the top of the screen vertically.





For example, here we see a distant view of an outline for a presentation about Pad++ itself.

Click on the next button to go on.



When you zoom in, the major headings become legible.

Click on the next button to go on.



When you click on a piece of text, Pad++ zooms in so that that text fills the screen horizontally and appears near the top of the screen vertically.



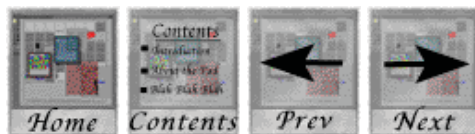
Interaction with the outline view

To move through a presentation shown in the outline view, you can simply click on successive pieces of text.

Alternatively, the 'n' and the 'p' keys move to the next or the previous item in the presentation using smooth animated zooms.

Of course, at any time you can use the mouse to pan and zoom to another part of the presentation - great if you want to quickly skip a few sections or go back to a previous section.

Click on the Next button below to read on.



Hypermedia presentations using outlines

Outline documents can include things other than text: images and interactive objects can be incorporated in the outline; you could even include a whole document nested within the outline, drawn at a smaller scale - here you see an HTML document and an image nested within the outline:



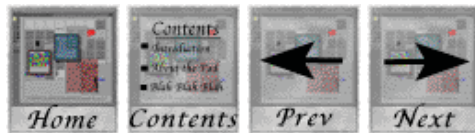
2. A file browser

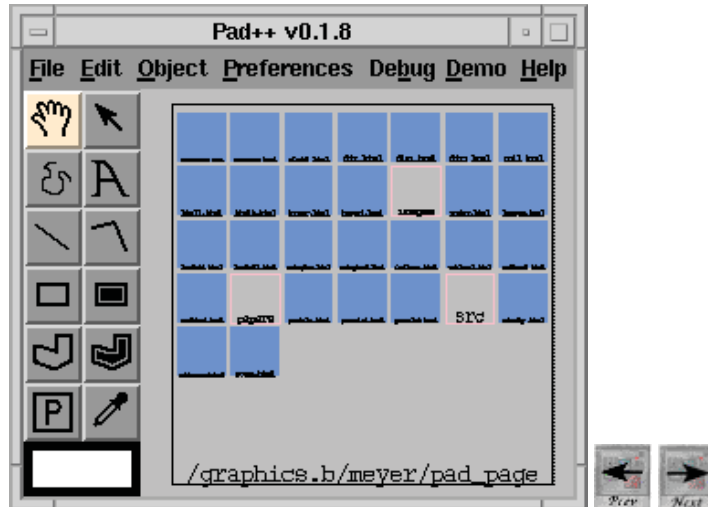
Many hierarchical structures are suitable for visualisation within Pad++. The second application we will look at is a file browser.

An important concept in the zoomable coordinate system is that objects have an absolute physical location. The metaphor is not one of viewing a small part of the data through a window, as in MS-Windows. Instead, all of the data is placed directly on the Pad++ surface, and you navigate through the data by panning and zooming. In this sense, Pad++ is a 2.5 dimensional virtual reality.

For example, in the Pad++ file browser, each file is shown as a small rectangle with a label.

Click on the Next button below to read on.

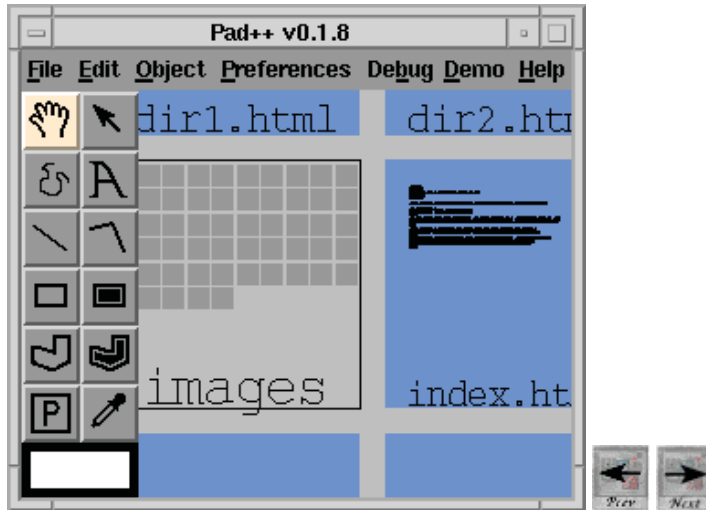




Above you see the file browser looking at some of the files that constitute this document.

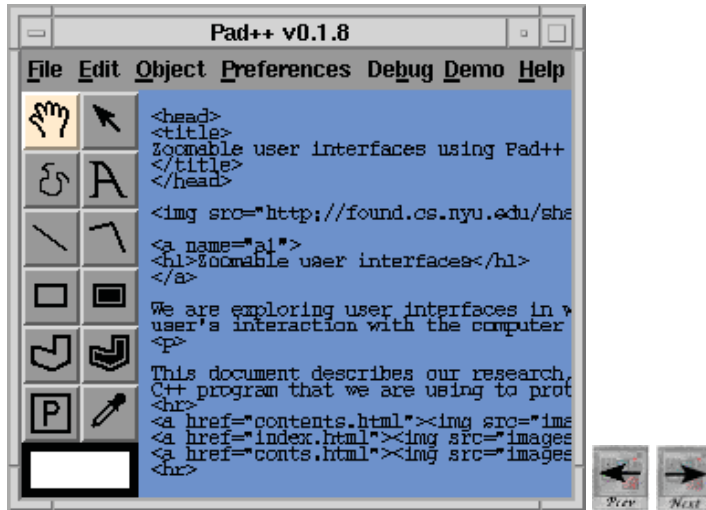
Colors are used to denote file type - text documents are shown using a solid blue rectangle, whereas the 'images' folder is shown as an unfilled rectangle outline.

Click on the next button to go on.



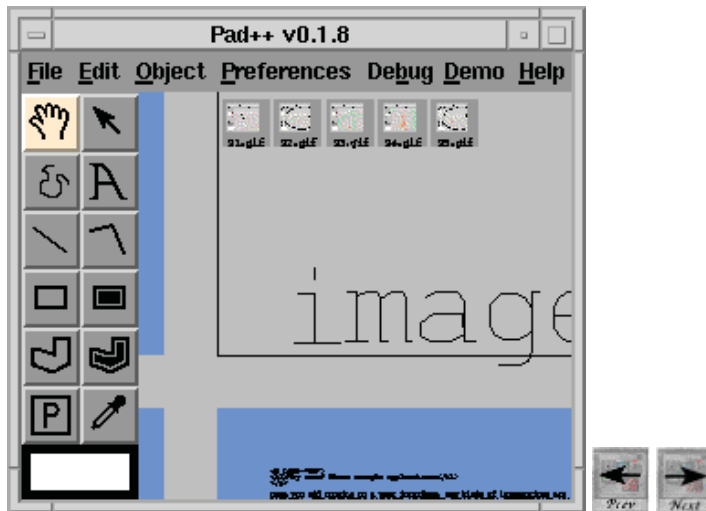
To see more detail, simply zoom in. Here, we have zoomed in for a closer view of the 'images' folder and the 'begin.html' document. At this distance, you can't make out individual pictures or words, but you can see the outlines of files within the 'images' folder and the shape of the text in the HTML file.

Click on the next button to go on.



To read the contents of the 'begin.html' file, zoom in further.

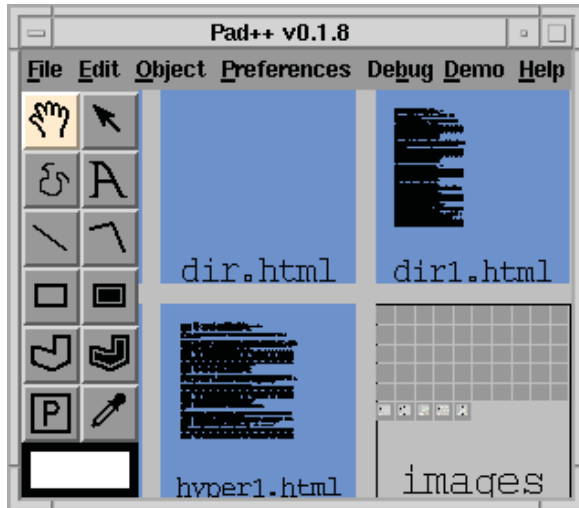
Click on the next button to go on.



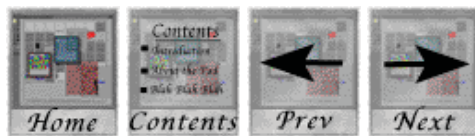
Now we have zoomed out and panned across.

You can see the pictures within the 'images' folder, and some of the contents of another file. Of course you could zoom in further to see the images in greater detail.

Click on the next button to go on.



Zoom out again. The file browser performs 'lazy loading'; it only loads the contents of the files that you zoom in on. When you zoom out again, you can see which documents you have seen and which are still unloaded.



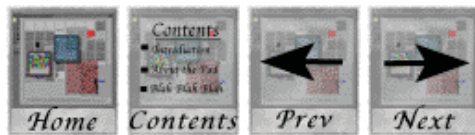
Semantic zooming in the file browser

The file browser illustrates what we call 'semantic zooming' - objects can render themselves differently when viewed at different sizes. So when a file is very small, it is shown only as a rectangle. A little larger and the label appears. Larger still and the contents are visible.

Through semantic zooming, many of the mode switches required in traditional windowing systems can be removed. The user does not have to double click on a file icon to switch into a mode where its contents are visible - instead they learn and reuse the simple principal that, to see more detail, you look more closely.

Lets move on to the third application in Pad++...

Click on the Next button below to read on.

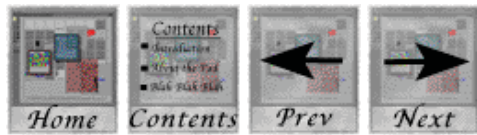


3. An HTML browser

A zooming version of Mosaic and Netscape? This is not such a strange idea. The third example Pad++ application we will look at is a simple Pad++ web browser.

Although the web browser we show here is still fairly primitive, you could imagine a much more advanced version. Does the idea of using a zoomable Web browser excite you? If so, why not email us at pad-comment@cs.unm.edu and tell us your thoughts.

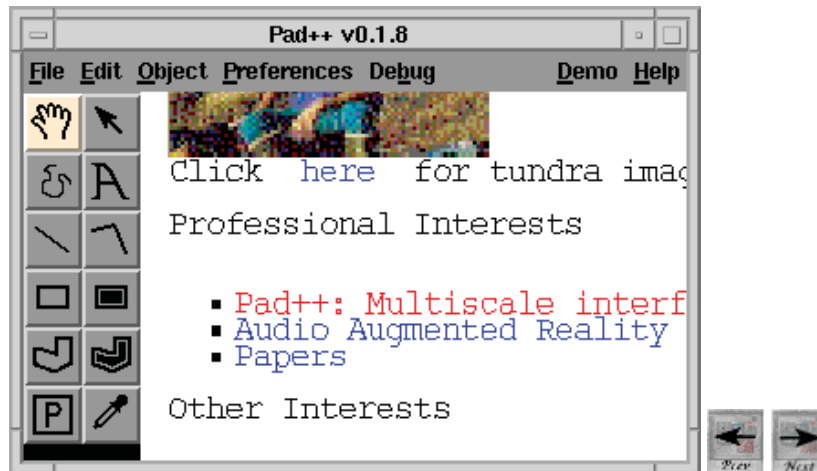
Click on the Next button below to read on.





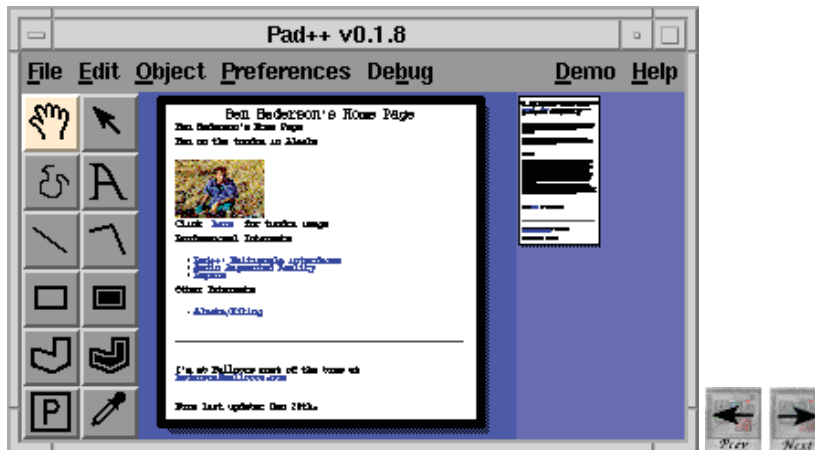
Here is a screen snapshot showing Pad++ displaying an HTML document.

Click on the next button to go on.



Here is a zoomed in view of the document. Hotwords are shown in blue - positioning the pointer over a hotword changes its color to red. In this snapshot the pointer was over the 'Pad++: Multiscale interfaces' link.

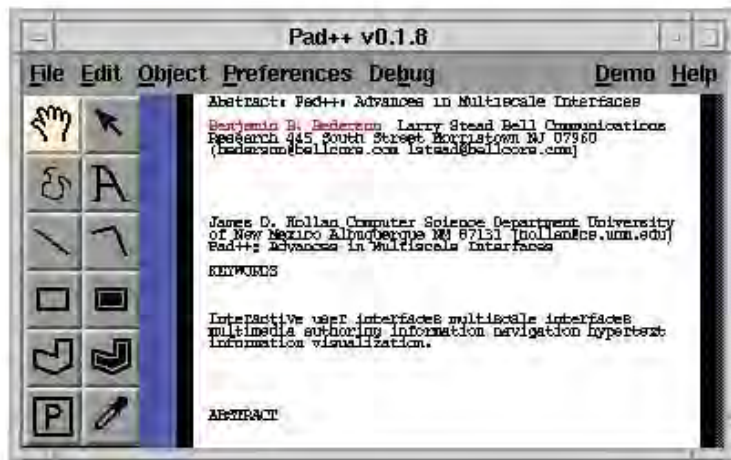
Click on the next button to go on.



When you follow a link, the relevant document is loaded into Pad++ and placed on the surface to the right of the original document, at a smaller scale. Here you can see the 'Pad++: Multiscale interfaces' document loaded beside the home page.

The Pad++ HTML browser will lay out sub-documents in two columns next to the parent document. Because Pad++ is zoomable, there is always enough space between those two columns for placing further documents reached from those sub-documents!

Click on the next button to go on.



Simply zoom in on the sub-document to read it.



Preserving history in the HTML browser

You can see the history of the user's interaction implicitly in the layout of the documents on the Pad++ surface. Here's what the Pad++ surface looks like after we've done a little browsing. Four documents were accessed via Ben's home page. From the first of these, another two documents were visited. Just zoom in to view the documents.



Other Pad++ features

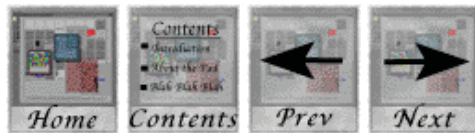
We have seen three example Pad++ applications. Now we will look at some other Pad++ features: portals, hyperlinks and magic lenses.

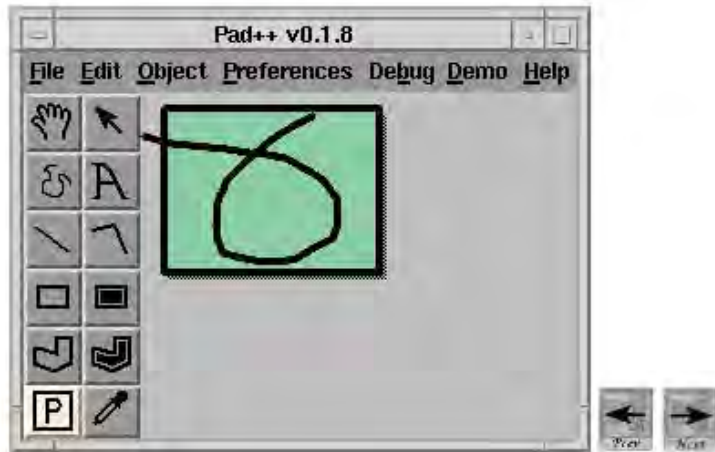
Portals - twists in Pad++ space

The Pad++ surface as an infinite zoomable space, with objects residing at specific x , y , $zoom$ coordinates within that space.

Sometimes, however, you want to be able to compare two distant locations, or to look at a zoomed out view of a document together with a close up view.

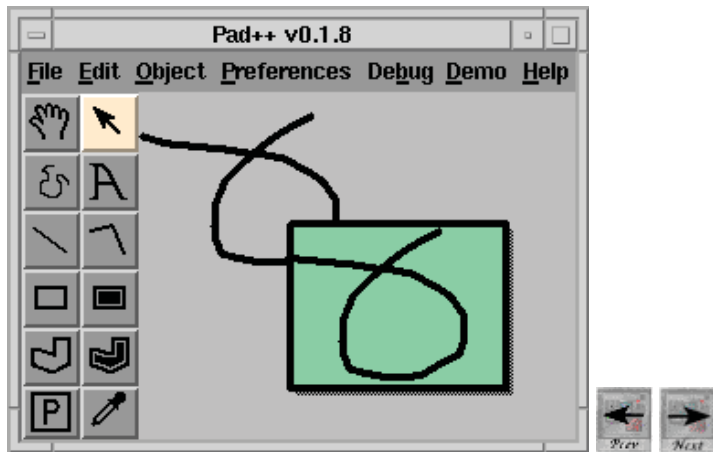
Portals allow you to do this - they introduce 'holes' in the Pad++ surface through which you look onto other regions of the Pad++ surface, or in fact onto other Pad++ surfaces.





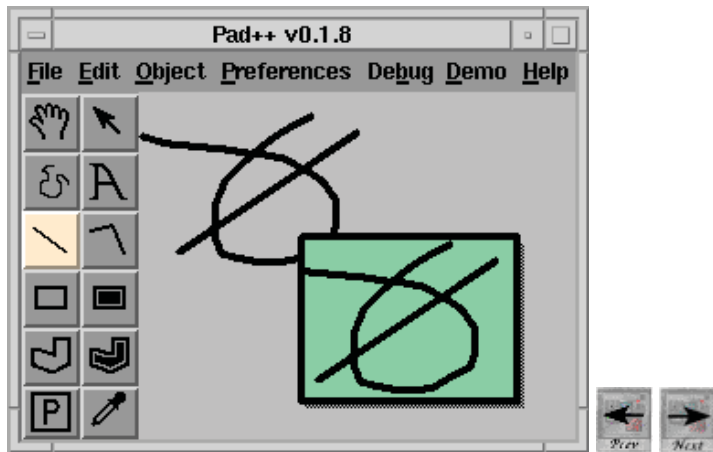
When you initially create a portal it looks like a colored rectangle with a drop shadow. Here we have created a portal over a squiggle on the Pad++ surface.

Click on the next button to go on.



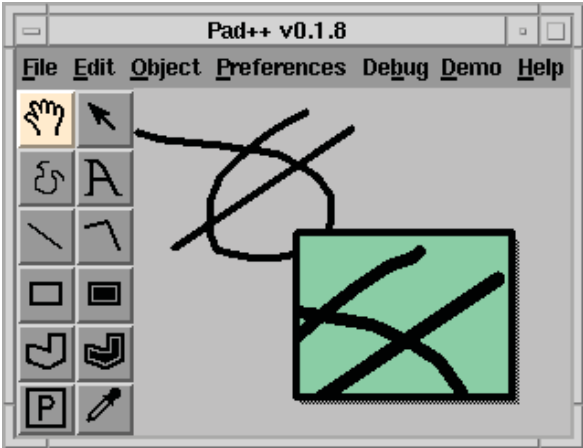
When you select the portal and move it away, it drags off a copy of what was underneath it.

Click on the next button to go on.

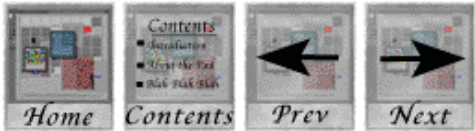


In fact, however, it isn't a copy - its the 'real thing' - its another view onto the same part of the Pad++ surface. So when you draw a second squiggle next to the first, it appears in two places... on the Pad++ surface, and within the portal's view of the Pad++ surface.

Click on the next button to go on.



Each portal has its own independant view - you can pan and zoom inside a portal without changing the view outside the portal. Here we see the portal looking at a zoomed in view of the squiggle and line.



Portals vs Windows

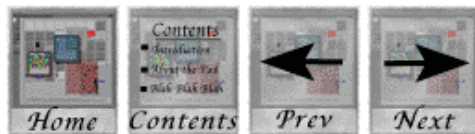
A portal is not like a window, which represents a dedicated link between a section of screen and a specific thing (eg: a Unix shell in X-Windows or a directory in the Macintosh Finder). A portal is, rather, a view onto the infinite Pad++ surface; links to specific items are established and broken continually as the portal's view changes.

Also, unlike windows, portals can recursively look onto other portals - you can get some pretty interesting effects with portals:



We will discover more about portals later. First, lets look at hyperlinks.

Click on the Next button below to read on.



Hyperlinks

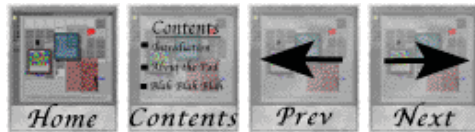
We have seen that, when you move the pointer over text within the [Pad++ outliner](#), the text changes color to red; clicking on the text causes Pad++ to pan and zoom until that text fills the screen horizontally and appears near the top of the screen vertically. The text is acting as a hyperlink to a location on the Pad++ surface.

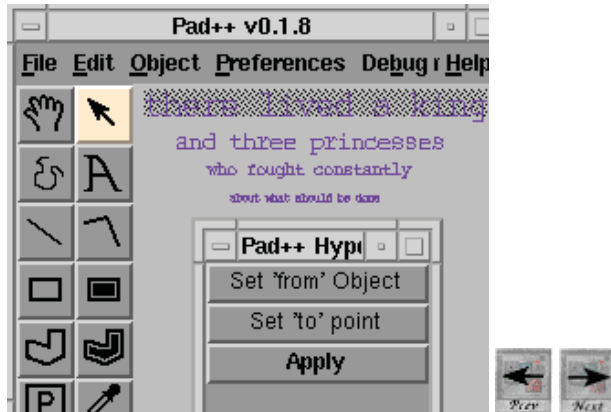
(Similarly, in the [Pad++ HTML browser](#), some words are shown in blue. They are 'hotwords' - positioning the pointer over the word causes it to change color to red. When you click on the word, the appropriate document is placed on the Pad++ surface.)

In Pad++, this concept is generalised. Any object can be turned into a hyperlink.

For example, suppose I am writing a zoomable narrative, and I want to make the text in the story active, so that users can click on lines of text to progress through the story. Read on to discover how to do this.

Click on the Next button below to read on.

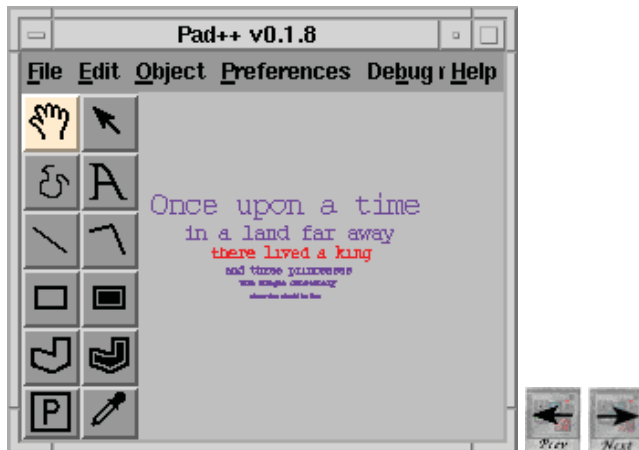




In this snapshot we are making the 'there lived a king' line of text a hyperlink.

From within the Pad++ draw application, bring up the 'Hyperlink' panel. Then set the 'from object' (the object you wish to make active), and the 'to point' (the place you want the hyperlink to take you to), then click on apply.

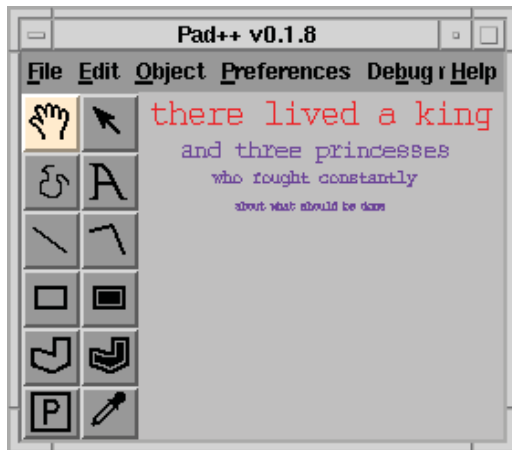
Click on the next button to go on.



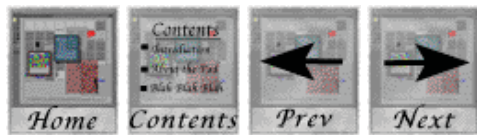
The text is now active. Here we have zoomed out and you can see the whole story.

The pointer is positioned over the 'there lived a king' line, and it has changed color to red to indicate that it is a hyperlink.

Click on the next button to go on.



Clicking on the 'there lived a king' line performs an animated pan/zoom to the location that the author specified.



Sticky objects

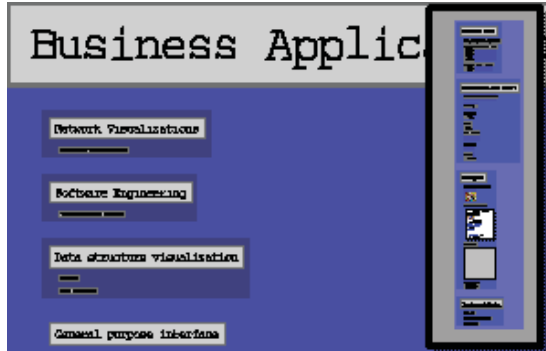
Pad++ objects can be made 'sticky' - so that they stick to the surface of the glass, rather than being on the Pad++ surface. When you pan and zoom the Pad++ surface, sticky objects remain stationary. In these snapshots, the 'Pad++' label has been made sticky, and does not pan and zoom with other objects:



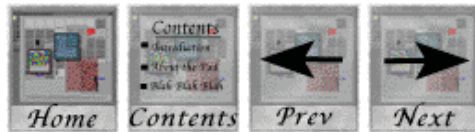
Sticky portals as indexes

By making a portal sticky and changing its view so that it looks at a hotword or an active document, you create automatic hyper-buttons and indexes which float above the Pad++ surface.

Here you can see an example of a portal acting as an index to the Pad++ outline: Clicking on text within the portal causes the main Pad++ surface to pan/zoom to show the text you clicked on, giving you rapid access to the outline.



Click on the Next button below to read on.



Portal filtering and 'magic lenses'

Portals can be used to filter what is seen through them. This concept goes by the name of 'Portal Filters' or 'Magic Lenses'.

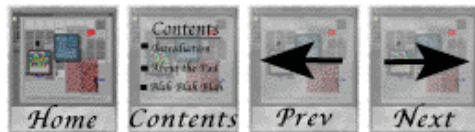
When an object is seen through a portal, the portal can communicate with the object during rendering to change how the object appears.

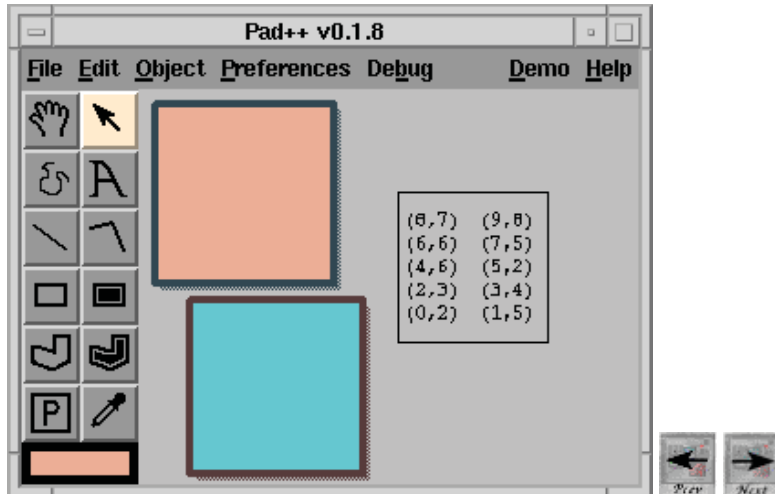
The portal can also filter user events (mouse clicks, etc.) which pass through it - either passing these events on unchanged, blocking them, or communicating with the object to decide how to interpret the event. So objects can behave differently when seen through portals.

By using these features, developers can create portals which act as magic lenses - lenses which change the nature of the data in unique ways. (the concept of portal filters and magic lenses was developed independently by Ken Perlin and David Fox in the original version of Pad and by Eric Beir, et. al. at Xerox PARC).

Lets look at some examples.

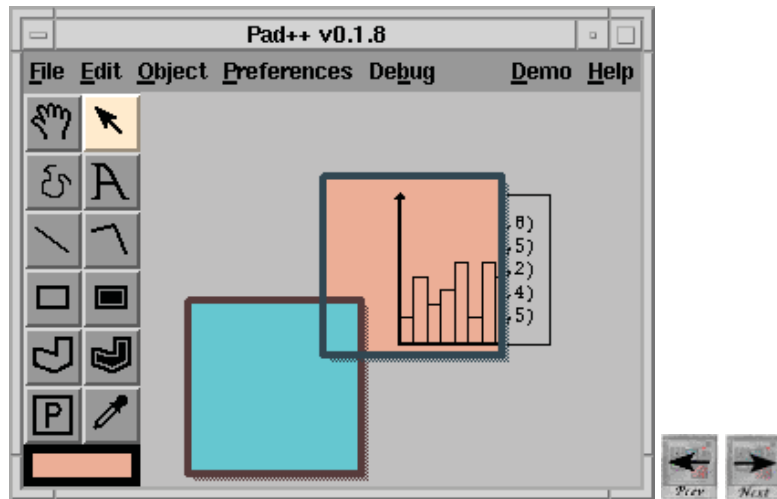
Click on the Next button below to read on.





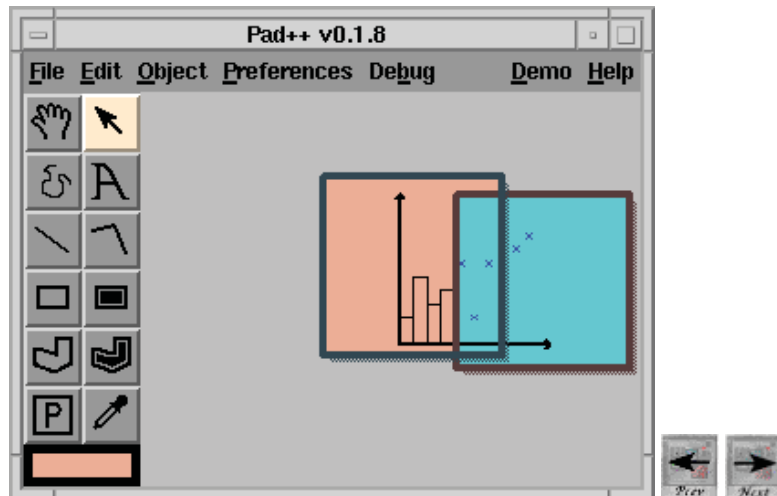
In the image above you can see a table of numbers and two portals.

Click on the next button to go on.



When you slide the orange portal over the table of numbers, it modifies how the table appears so that it uses a bar chart view.

Click on the next button to go on.

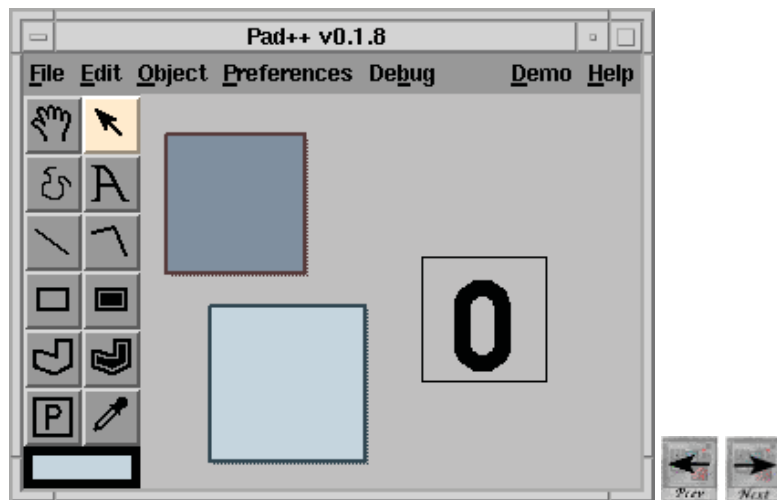


Slide the blue portal in place, and you see the table of numbers as a scatter plot.

Lenses such as these could make a good teaching aid, since they show students alternate views of the same data.

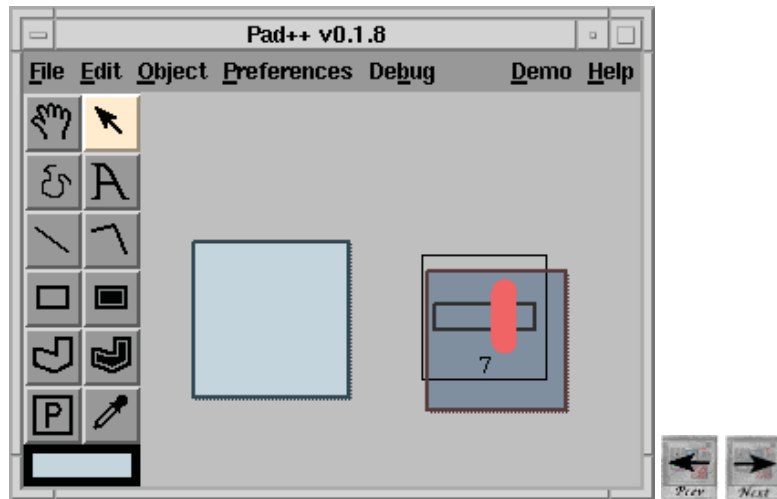
Lets see another example.

Click on the next button to go on.



In this example, the 0 digit shown above is a simple digital counter. When you position the pointer over the number and hit a number key between 0 and 9, the counter changes to show that number.

Click on the next button to go on.

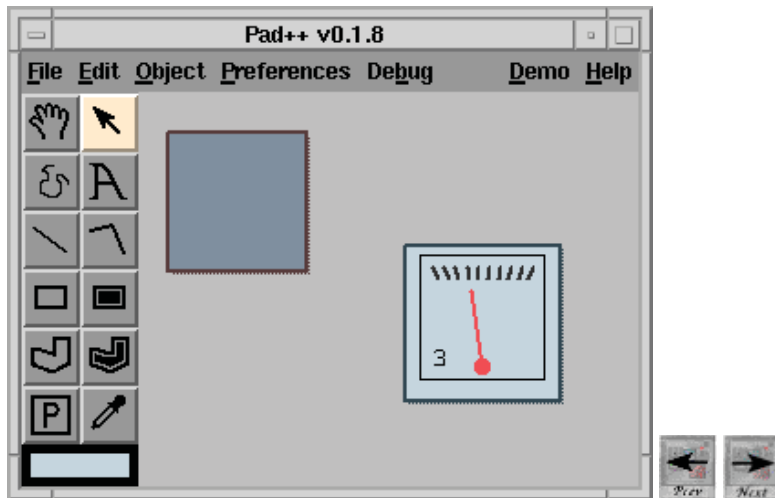


However, some people prefer to use a slider to set numeric values.

Here, the dark gray lens has been placed over the number, and the user has clicked and dragged the slider handle to change the number to 7.

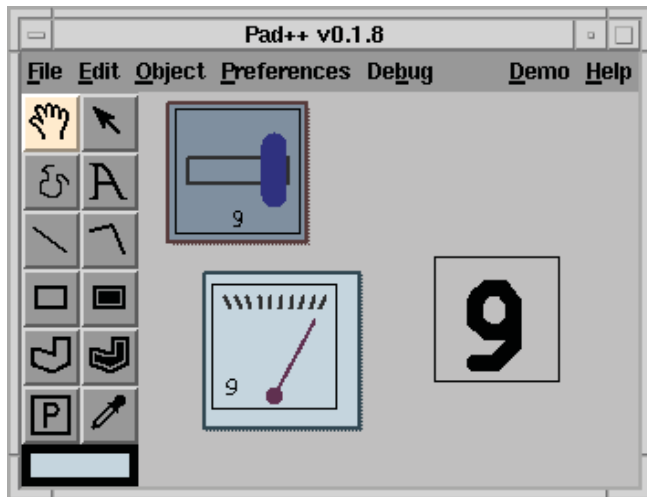
This shows how portals can change how an object behaves as well as how it appears.

Click on the next button to go on.

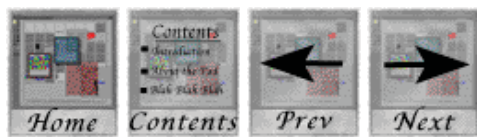


You may prefer to see the number as a gauge. Use the light gray lens. Here we are setting the number to 3.

Click on the next button to go on.



With a little simple trickery, all three representations can be used at the same time - you can choose which version to interact with.



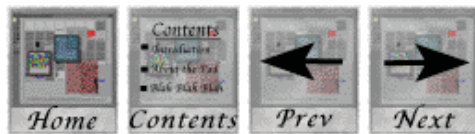
Other uses for lenses

Whilst the example shown here are relatively trivial, you could create a much more complex lens.

Lenses are used for rendering and interacting with data in specialised ways. You could for example create a translation lens, which shows a piece of text in an alternate language.

More sophisticated lenses might be used for editing text, or for painting, or for modifying bitmap data. Instead of constantly switching between large applications and using the Import/Export options to move data between the applications, you simply pick up the relevant lens and slide it into place.

Click on the Next button below to read on.

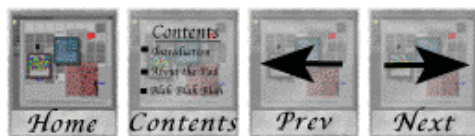


Conclusion

In Pad++ there is a sense of peripheral awareness. When objects are small or off to one side, you can still see them, and still make out a little of their contents, though they are not shown in all their detail. When you look directly at them and zoom in, all of the detail becomes available. Hence the Pad++ metaphor offers a new route for tapping into our natural spatial and geographic ways of thinking.

This metaphor is more powerful than the traditional (and now dated) view of a computer screen as a collection of small windows through which you peer at your data.

Click on the Next button below if you want to read answers to frequently asked questions about Pad++.



Frequently asked questions

This document contains a list of answers to common questions about Pad++ and Zoomable User Interfaces.

Questions about the Pad++ consortium

- [Where did Pad++ come from?](#)
- [Who is working on Pad++?](#)
- [What research directions are you following?](#)
- [How do I contact the Pad++ group?](#)
- [How do I get a Pad++ license?](#)
- [How do I get on the Pad++ mailing lists?](#)

Questions about Pad++

- [What other online info is there on Pad++ or Zoomable interfaces?](#)
- [Is Pad++ available?](#)
- [What platforms does Pad++ run on?](#)
- [Why does the Windows version run slower than the Unix version?](#)
- [Is there a Pad++ ftp site?](#)
- [What are the plans for the Web browser?](#)
- [Are you going to port Pad++ to Java?](#)

Questions about the approach

- [Its all very well, but why isn't Pad++ 3D?](#)
- [How does Pad++ relate to traditional windowing systems?](#)

Technical questions about Pad++

- [What does the Pad++ code consist of?](#)
- [How do you get Pad++ to zoom so quickly?](#)
- [Pad++ dies as it starts up on Windows95/NT. Why?](#)
- [Building Pad++ on Solaris with gcc 2.8.1 hangs. Why?](#)

Technical questions about Tcl/Tk that Pad++ users have asked

- [Why does my Tcl socket code hang when using Pad++?](#)

Questions about the Pad++ consortium

Where did Pad++ come from?

Ken Perlin at New York University came up with the initial zoomable surface concept. He and David Fox implemented the first versions of Pad (described in the 1993 SIGGRAPH paper available [here](#)), which were the precursors to the current implementation, Pad++. Perlin and Fox's original work was supported in part originally by NYNEX, and then NSF.

Pad++ was designed by Ben Bederson and Jim Hollan while at Bellcore, and implemented by Ben Bederson. They both moved to the University of New Mexico where they developed the bulk of Pad++ in collaboration with Jon Meyer and Ken Perlin at NYU.

Now, Ben Bederson has moved to the University of Maryland, College Park where he is continuing responsibility for the further development of Pad++. Jim Hollan moved to the University of California, San Diego where he is building applications using Pad++.

Who is working on Pad++?

Pad++ research and development is supported in part by DARPA grant #N660011-94-C-6039. Work is being carried out at the University of Maryland, University of California, San Diego, and at the NYU Media Research Laboratory. Much of the initial work on Pad++ was done at the University of New Mexico.

Here are the links to folks currently working on Pad++:

- **University of Maryland, HCIL**
[Benjamin Bederson](#)
[Allison Druin](#)
[Tammara Combs](#)
- **Univ. of California, San Diego, Cognitive Science Dept.**
[James Hollan](#)
- **University of New Mexico, CS Dept.**
[Jason Stewart](#)
- **NYU Media Research Laboratory**
[Ken Perlin](#)
[Jonathan Meyer](#)

And here are the folks who have worked on Pad++ in the past:

- **University of New Mexico, CS Dept.**
[Hugh Bivens](#)
[George Hartogensis](#)
[David Proft](#)
[Laura Ring](#)
[David Rogers](#)
[David Vick](#)

Ron Hightower
Mohamad Ijadi
David Thompson
Ying Zhao

- **The NYU Media Research Laboratory**

David Bacon
Troy Downing
David Fox
Athomas Goldberg
Noah Wardrip-Fruin

- **Bellcore**

Larry Stead

What research directions are you following?

The Pad++ group is developing Pad++ in many ways, such as:

- Creating Pad++ GUI components.
 - Developing an internet based distributed Pad++.
 - Performing Pad++ usability studies.
 - Developing Pad++ applications for understanding large datasets.
 - Porting Pad++ to Windows95/NT.
 - Developing easy to use authoring tools.
-

How do I contact the Pad++ group

You can contact the Pad++ group at the following email addresses:

- pad-info@cs.umd.edu - Send your comments and information requests here
 - pad-bug@cs.umd.edu - Report bugs here
 - pad-chat@cs.umd.edu - Send questions/comments to all Pad++ users that have asked to be on this list
-

How do I get a Pad++ license?

For non-commercial (education, research, or in-house) use, all you need to do to use Pad++ is to fill in the online [registration form](#). If you are interested in a commercial license, you should contact us at pad-info@cs.umd.edu. The Pad++ Consortium currently consists of UNM and NYU, and license can be negotiated from either institution.

How do I get on the Pad++ mailing lists?

When you download Pad++, there are two checkboxes to add you to the mailing lists. Or, you can send mail to pad-info@cs.umd.edu, and we'll add you manually.

There is an *announce* list which has very low volume. This list is just for us to send you mail about new Pad++ releases and other important information. This is a private list, and others can not use it, so

you don't have to worry about getting much unsolicited mail. There is also a *chat* list which has moderate volume. This list is for the Pad++ user community to communicate with us and each other about ideas, solutions, requests, etc.

Questions about Pad++

What other online info is there on Pad++ or Zoomable interfaces?

There are several online papers about Pad++ and related issues [here](#)

Is Pad++ available?

Yes it is. You can register online and download it [here](#).

What platforms does Pad++ run on?

Pad++ runs on just about any Unix platform. We have tested it on PCs running Linux, SGIs, and Suns (SunOS and Solaris).

Pad++ now also runs on Windows95/NT. This port is still early, and the Windows version is not very reliable. Also, it is slower because it is currently implemented with Tk's X emulator.

Why does the Windows version run slower than the Unix version?

The Windows version of Pad++ runs slower than the Unix version because we are using the Tk X emulator. In a future version, we will write a native Windows renderer, and expect that it will be at least as fast as the Unix version.

Is there a Pad++ FTP site

No, all public information about Pad++ is at this web site.

What are the plans for the Web browser?

We built a prototype Web browser in Pad++ in 1995. At that time, HTML was quite simple, and the idea of building our own better browser was quite tempting. We build what was at the time a cutting edge browser - with HTML 1.0. Since then, the web has advanced greatly, and we realized that we

didn't want to get in the business of competing with the major Web browser companies.

Instead, we are taking advantage of the new features of HTML by building a new generation Web browser that piggybacks on existing commercial ones. The new Pad++ Web browser, called PadPrints, monitors what the browser is doing and what pages are being visited. It builds a map showing your history of interaction with Web. You can use your commercial browser normally, but at any time, you can go to PadPrints in a separate window to control the browser.

Are you going to port Pad++ to Java?

We are looking into connecting Pad++ to Java so that it will be possible to write zooming applications entirely in Java using the C++ core. However, this work still has a long way to go.

Another approach would be to build Pad++ entirely in Java. We have experimented with this, and it appears that at least for the time being, Java is too slow, and the AWT graphics package is not powerful enough. Perhaps when Java compilers are common, and Java2D is commonly available (and well-implemented), it will be possible, but that is still in the future. However, we have created a simple zooming Java applet to give a flavor of what the interaction feels like. See the link on the [home page](#).

Questions about the approach

Its all very well, but why isn't Pad++ 3D?

There are some very attractive 3D systems out there (the Information Visualizer is one of them). However, although they may look impressive, 3D systems are typically hard to navigate using current display and pointer technologies. They also require high levels of computing resources. Its not obvious that a 3D interface is appropriate for a handheld device or for a cheap home computer.

Pad++, on the other hand, can be implemented very efficiently using a small home computer, and is easy to navigate using a mouse.

How does Pad++ relate to traditional windowing systems?

There are two answers to this - choose one you like:

1. Pad++ can be seen as a mechanism for creating applications which have a zooming component. For some applications, a zoomable interface may be more appropriate than a more traditional windows based approach - for example, zooming seems to be particularly effective for applications which need

to present large amounts of data in an intuitive and understandable manner. This especially true for data that is hierarchical in nature as information that is deeper in the hierarchy can be represented as being smaller on the Pad++ surface.

2. Pad++ could also be viewed as a way to implement an alternative windowing system based on zooming. Your whole desktop could be zoomable. This seems especially attractive for systems which have small screens, such as handheld computers (i.e. PDA's).

Technical questions about Pad++

What does the Pad++ code consist of?

Pad++ consists of a C++ core that provides zoomable objects. Applications can be written in C++ (this API is still under development), or with [Tcl/Tk](#), a scripting language that implements the zoomable draw application which you have seen.

Pad++ runs on most unix machines including PC's running Linux, on workstations from Silocon Graphics, IBM and Sun, and probably on most other UNIX-like systems that support X11. Pad++ also runs on Windows95/NT, but this port is still slow and unreliable.

How do you get Pad++ to zoom so quickly?

In order to keep zoom animations fast, Pad++ employs several tricks:

Spatial Indexing

Pad++ uses an R-Tree internally to quickly determine which objects are visible in a given view.

Level of Detail

When things become too small to see, Pad++ ceases to render them. When Pad++ starts to get too slow, it renders medium sized things in a more ugly fashion. When the system is idle, things that have not been rendered fully are *refined* - see below.

Refinement

When trying to achieve a high frame rate, Pad++ use a reduced level of detail. Then, when the system is idle, successive refinements are performed to increase the level of detail.

Adaptive render scheduling

The system monitors render times, and adapts the rendering algorithm to maintain a constant frame rate during zooms and pans.

Pad++ dies as it starts up on Windows95/NT. Why?

There is a bug in the Windows95/NT version where it dies if there is a **space** in any of the directory names that Pad++ is installed in. So, if you installed Pad++ in the '*Program Files*' directory, it will die with this behavior. Try re-installing (or just moving Pad++) in the top-level directory, and it should work. Sorry about this - we'll try and fix this in the next version.

Building Pad++ on Solaris with gcc 2.8.1 hangs. Why?

We've had reports that building Pad++ on Solaris 2.5 with gcc 2.8.1 hangs when compiling generic/object.cpp. A workaround is to remove the -fPIC flag from the Makefile. This flag is only necessary when compiling shared libraries, but isn't necessary when just compiling an executable, so you should be able to remove it safely.

Another approach is to configure with:
configure --disable-load
and then add '-ldl' at the end of the link line

Why does my Tcl socket code hang when using Pad++?

When Tcl 7.6 performs a socket read in "line buffered" mode, it reads only until the new-line character. If the writer sends a zero-terminated string, Tcl 7.6 leaves the zero byte in the buffer. The next read, then, returns a string that begins with a zero byte; we interpreted this incorrectly as a read of a null string, when in fact it was a long string following a zero byte. This problem does not occur with Tcl 8.1, the latest release.