Fisheye Menus

Benjamin B. Bederson
Human-Computer Interaction Lab
Institute for Advanced Computer Studies
Computer Science Department
University of Maryland, College Park, MD 20742
+1 301 405-2764
bederson@cs.umd.edu

ABSTRACT
We introduce "fisheye menus" which apply traditional fisheye graphical visualization techniques to linear menus. This provides for an efficient mechanism to select items from long menus, which are becoming more common as menus are used to select data items in, for example, e-commerce applications. Fisheye menus dynamically change the size of menu items to provide a focus area around the mouse pointer. This makes it possible to present the entire menu on a single screen without requiring buttons, scrollbars, or hierarchies.

A pilot study with 10 users compared user preference of fisheye menus with traditional pull-down menus that use scrolling arrows, scrollbars, and hierarchies. Users preferred the fisheye menus for browsing tasks, and hierarchical menus for goal-directed tasks.

Keywords
Fisheye view, menu selection, widgets, information visualization.

INTRODUCTION
The concept of a "fisheye" distortion in a computer interface to present detailed information in context has been around a long time. Spence & Apperley introduced the idea in 1982 [24]. Furnas then discussed the cognitive aspects of how people remember information [7]. Several researchers then applied fisheye distortion to a broad variety of applications [4, 15, 25]. Several variations of the fisheye technique have been explored. They have been used in one dimension for word processing [9], access to time [12], and for long lists [13, 14]. They have been used in two dimensions for tables [17], graphical maps [20] and space-scale diagrams [8]. They have even been used in three dimensions for document browsing [19]. Some applications of fisheye distortion techniques have been carefully evaluated, often finding a significant advantage to fisheye views [5, 11, 21].

However, despite the careful investigation of fisheye view distortion techniques, and their application to a broad set of complex tasks, fisheye views have never been applied to the mundane challenge of ordinary menus. This paper applies standard fisheye techniques to menus in Graphical User Interfaces with the goal of improving performance in user's ability to select one item from a long list.

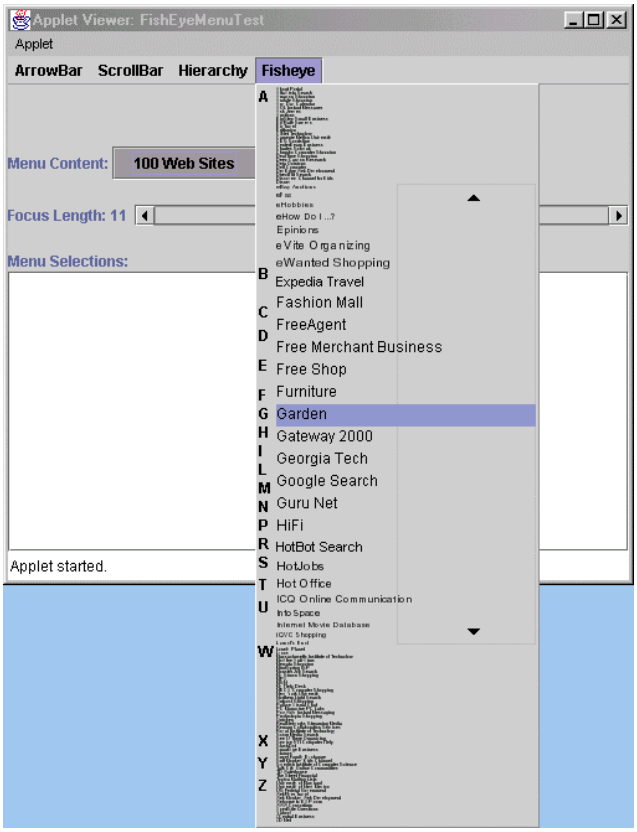

Figure 1: A screen shot of the fisheye menu in use. This shows 100 web sites taken from the most popular list of PC Magazine.

Selecting items from menus is another well-studied area, and the trade-offs of menu design are well understood [10, 16]. Menu design has become quite standard with well-grouped menu items in consistent locations using common names. This is appropriate for carefully designed applications where every element of the menus can be chosen in advance.

However, with the introduction of the Web and e-commerce applications, it is becoming increasingly common to use menus for selecting data items, as opposed

to selecting operations. For example, menus are used to select from a long list of fonts, to select one state out of 50, to select one country out of 250, or to select a web site from a list of favorites.

It was this last example that motivated the application of fisheye views to menus. Managing ones favorite locations on the web is an important application of web browsers, but one study showed that most web browser users don't put more than about 35 items in their favorite lists before resorting to using hierarchies [1]. While hierarchies certainly help to organize information, this study found that while some people used hierarchies, many stopped adding new favorites altogether. The user interface for managing favorites may contribute to this. Since web browsers use pull-down menus to store favorites, and since these menus don't work very well as the number of elements within the menu grows, it is not surprising that people don't put more than that many items in the menus before using hierarchies. Some researchers have looked at alternative interfaces for managing web favorites [18], but they have not yet made it into commercial products. Also, those approaches are fine-tuned to web favorite organization, and may not apply very well to other menu selection tasks.

Selecting data items from menus is different than selecting functions because the data items in the menu are likely to change from use to use, and there are typically many more data elements in a menu than there are in functional menus. In addition, since the user is not as familiar with the menu, it is more likely that they won't know the exact text of each item. Thus, supporting browsing as well as searching is important. The length of the menu is crucial in determining usability. It takes users a time proportional to the location of an item in a menu to access it [6, 22]. However, the real problem comes with menus that have more items than fit on the screen. AlphaSliders are one approach for selecting textual items from a long list in a small space [2]. However that approach only displays one item at a time, and does not fit into the pull-down menu metaphor.

The existing approaches to selecting from one of many displayed items in a long list are limited. There are three commonly used approaches which are to use scrolling arrows at the top and bottom of the list, to use hierarchical "cascading" menus to make the list smaller, or to use scrollbars. Let us look at each of these approaches in more detail.

Standard GUI toolkits today provide support for long pull-down menus by adding small scrolling arrows to the top and bottom of the list if the entire list doesn't fit on the display. When the user clicks on those arrows, the list is scrolled up or down. Each toolkit implements these arrows differently, some having fast scrolling if you hold the arrow down (Microsoft MFC), and some slow (Swing). Some automatically scroll when the mouse is just placed over the arrows without clicking (Internet Explorer). However, in any case, the user is required to first move the mouse to the arrow, and then scroll until the desired element becomes visible. An additional, but uncommon problem is that if the menu is scrolled too far, the mouse must be moved to the arrow on the opposite side of the menu, and the user must then scroll in the other direction.

A common alternative to long lists is to use hierarchical "cascading" menus. This works by having the application developer, or sometimes the user, organize the menu elements into groups. Then, one entry that represents each group is placed in the menu. When the user selects that group element, the members of the group are displayed in a second menu off to the side. This approach solves the problem of physically navigating a long list, but replaces it with a new problem of requiring the user to know what group the desired element is in. If the user knows the hierarchy structure well, then this approach works. However, if the user does not know the hierarchy structure well, then the user must look in each group, which is potentially time consuming. Typical applications with stable menu structures regularly use hierarchical cascading menus because presumably the user will rapidly learn where each element belongs. However, it is uncommon in practice to find hierarchical menus that are used for organizing data driven menus.

Finally, the last common solution for managing long menus is to use a scrollbar that controls the portion of the menu that is visible. This seems like an excellent approach because it gives fixed time access to menus of any length unlike the more common scrolling arrows, which takes time proportional to the menu length. However, while scrollbars are commonly used in dialog boxes, they are rarely if ever used in pull-down menus. Perhaps this is because current toolkits do not provide this as a default behavior, although it is possible to implement it with some toolkits.

In addition to these visualization methods, nearly all toolkits support keyboard shortcuts for selecting menu items. There are often modeless shortcuts (such as Ctrl-C for "Copy") that select a menu element throughout the application, even when the menu is closed. In addition to those shortcuts, the keyboard can be used to select items in the menu when it is open. Developers can either specify which key should apply to each item by specifying a "mnemonic", or if it is left unspecified, the first character of the item is used. Thus, in an alphabetically sorted list, pressing any key will jump the cursor to the first item starting with that letter. Pressing it again will move to the next item starting with that letter, and so on.

These keyboard accelerators are very powerful as they bypass some of the shortcomings of the mouse-based interaction techniques just described. They give users direct access to either the target element, or at least to the general area if there is more than one element sharing the mnemonic. However, despite their power, many users do not use them at all. Some users are not aware of them, but others are aware of them and choose not to use them anyway. Perhaps this is because their hand is already on the

mouse and takes too long to reacquire the keyboard, or perhaps they don't know the keyboard well enough to justify searching for the right key. Or they may not know the exact text and actually are browsing the menu. And finally, some users may just not like using the keyboard when interacting with menus. People that only use the mouse for selecting menu items are likely to be the largest beneficiaries of fisheye menus.

FISHEYE MENU DESIGN ISSUES

We offer a new solution to the problem of menus that have more items than fit on the screen by using a fisheye view to display the menu elements. In fisheye menus, all of the elements are always displayed in a single window that is completely visible, but the items near the cursor are displayed at full size, and items further away from the cursor are displayed at a smaller size. In addition, the interline spacing between items is also increased in the focus area, and decreased further away from the focus area. In this manner, the entire list of items fits on a single screen. The items are dynamically scaled so that as the cursor moves, a "bubble" of readable items moves with the cursor (Figure 1). A fisheye menu applet can be found at http://www.cs.umd.edu/hcil/fisheyemenu.

The fisheye menu uses all the available screen space, and will calculate a distortion function so that the menu items always just fill the menu. There are two principal parameters of the fisheye menu that the application developer can control: maximum font size, and focus length. As with traditional menus, the designer can specify the font size, which for the fisheye menu translates in to the maximum font size, since some elements are rendered smaller. However, the designer can also specify the desired focus length. This specifies the number of items that are rendered at maximum size near the cursor.

The focus length parameter is important because it controls the trade-off between the number of menu items at full size versus the size that is used to render the smallest items. The fisheye menu dynamically computes the distortion function based on the available space and these input parameters. So, if the focus length is set to a large number (i.e., 20), then this will push the peripheral items to be very small, and as the user moves the cursor, there will be a lot of distortion. If, however, the focus length is set to a small number (i.e., 5), then there will be more room for peripheral items and they will all be a bit larger. Figure 2 shows this trade-off.

Alphabetic Index

A fundamental characteristic of the fisheye menu is that many of the menu items are too small to read at any given position. However, since it is common to organize menu items alphabetically for data menus, we can encourage this organization for fisheye menus without undue burden. Then, users can use their alphabetic knowledge to move the cursor to the area they expect the item to be at, thus bringing that portion of the menu into focus at which point they can read the menu items and select the particular item

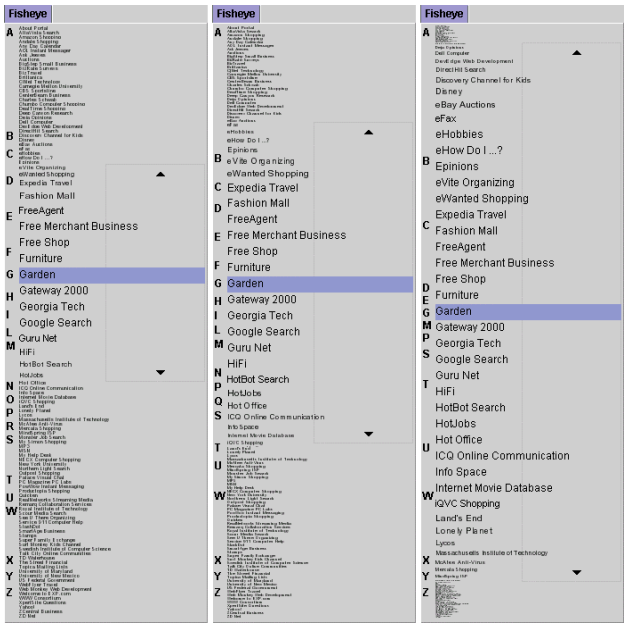

Figure 2: The same menu of 100 items displayed with varying focus lengths (7, 12, and 20). There is a fixed maximum font size.

they want. This is similar to how people use telephone directory books. Despite the fact that items are listed sequentially in the phone book, people use their alphabetic knowledge to jump to the portion of the phone book where they expect the item they are looking for to be. They then see where they actually are, and fine-tune their search.

This telephone book analogy guides the design. One of the reasons people can find items in telephone books so quickly is that telephone books have index information at the top of every page specifying in a large clear font what information is on that page. These indices allow users to just look at the indices while looking for the right page, and then look at the content when they have found the page they are looking for. It has been shown that indexes can decrease search time with lists [3].

We designed the fisheye menus to have an alphabetic index with the goal of making it easier for users to target the portion of the menu that contains the item they are looking for. The alphabetic index appears on the left side of the menu. Each letter of the alphabet for which there is room is displayed in the specified maximum font size.

The index letters are positioned so that when the pointer is moved to the same vertical position as an index letter, the first item starting with that letter will be just under the mouse pointer. This provides the user with the ability to rapidly move to the general area of the list they are targeting.

This is our second design of the index letters. The first design always positioned the letters at the current position of the first item starting with that letter. Thus, as the fisheye focus changed, the index letters would move around, following the items. This turned out to be

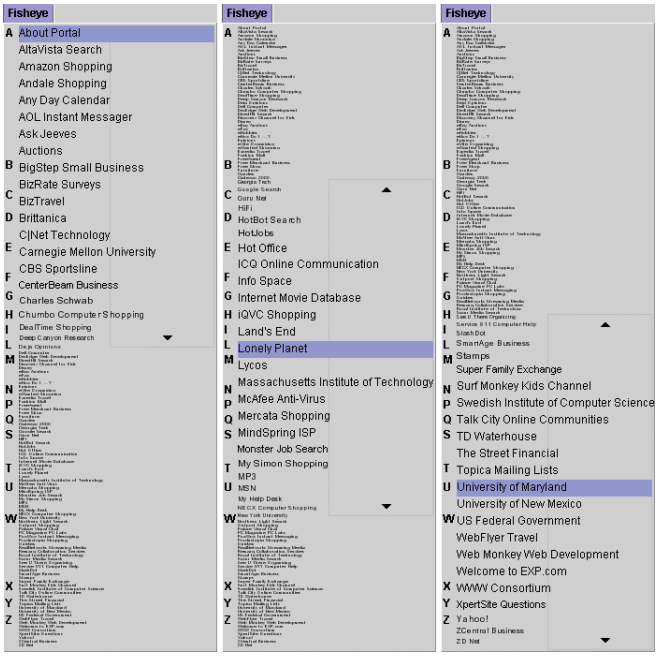

Figure 3: The same menu displayed with the cursor at three positions.

distracting and not useful. By the time a user moved the pointer to the position an index letter was at, that index letter would have moved (since the focus and thus item positioning would have changed.) We quickly realized the value of the index letters was to inform pointer motion, and shifted to the current stable design described above. Figure 3 shows the fisheye menu at different focus points.

High-Resolution Selection (Focus Lock Mode)

One difficulty with the fisheye menu mechanism as described so far is that small mouse movements result in a change of fisheye focus. With traditional menus, the mouse must move over the full height of a menu item to change the focus to the next item. However, with fisheye menus, the amount the mouse must move to go to the next item is equal to the *smallest* font size in the menu. This is a fundamental result of the fisheye algorithm since all of the menu items must be selectable by pointer movement in the fixed vertical space of the menu.

This is a significant liability because despite the fact that the focused elements are large and plainly readable, they are difficult to select.

We overcame this problem by offering a "focus lock" mode to the fisheye menu. Users operate the menu as described above until they get near the item of interest. They then move the pointer to the right side of the menu, which locks the focus on the item the cursor is over. Then, when users move the pointer up and down, the focus stays fixed, but individual menu elements can still be selected. The focus region on the right side of the menu gets highlighted to indicate that the menu is in focus lock mode.

Further, if the pointer is moved above or below the focus region (staying on the right side of the menu), the focus area is expanded. Eventually all of the menu items become full-size and thus easy to select. But, of course, not all of the items are visible anymore as the ends get pushed off the screen as the focus area is expanded. Since the menu layout is quite different in focus lock mode, the index characters become inaccurate, and so they are faded out as the focus area is expanded in focus lock mode.

If users decide to continue looking in a different portion of the menu, moving the pointer back to the left side of the menu turns off focus lock mode, and the menu returns to regular behavior. This focus lock approach to high-resolution selection within a fisheye view solves the resolution problem at the cost of a small mouse movement.

We considered several alternative approaches to entering the focus lock mode. We first tried using the right button, but gave that up as it seemed too unlikely that users would discover it on their own – especially since it did not follow the standard Windows model of pressing the right button for a context-sensitive menu. And, of course, it would not work at all for systems without a second mouse button. We also considered using the speed of the mouse to determine the focus mode, but that seemed to be too unpredictable by users. Also, an earlier study of the AlphaSlider confirmed this intuition [2].

We ended up with the current design, which offers an affordance for the focus lock feature. There is a subtly shaded box on the right side of the menu that moves up and down with the focus. This was intended to draw user's attention to the right side of the menu. In addition, the two small arrows on the right side are intended to suggest to users that they can move the pointer up and down in focus lock mode. When the pointer is moved towards the arrows,

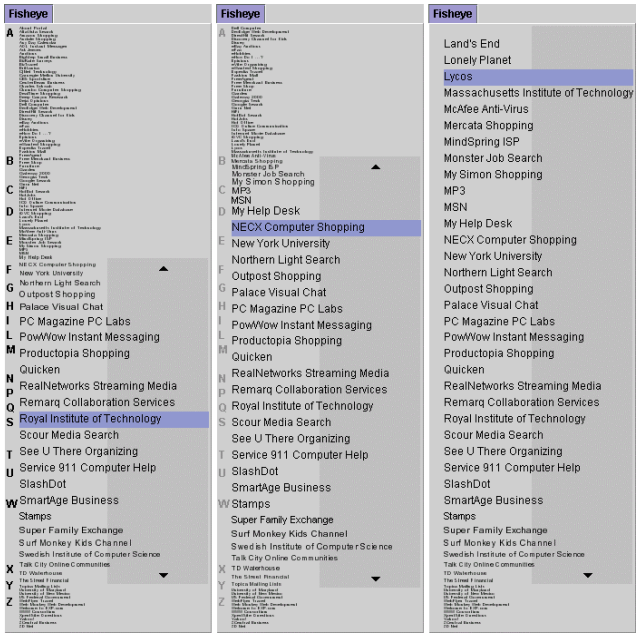

Figure 4: A fisheye menu in focus lock mode whose focus area is being extended upwards

the focus area is extended, and the arrows move accordingly. The users can thus discover that the focus can

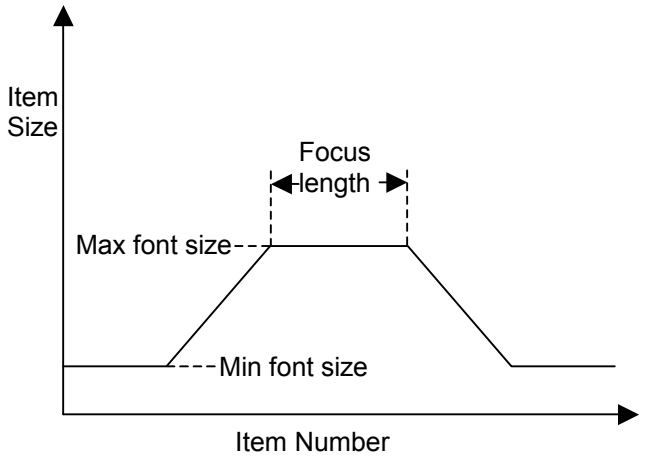

Figure 5: The basic Degree of Interest function used for the fisheye menu.

be extended. Figure 4 shows the focus lock mode with the focus area being extended upwards.

IMPLEMENTATION

The fisheye menu is a drop-in replacement for Java's standard "JMenu" component in the Swing GUI toolkit. This new widget, called FishEyeMenu, is written in Java 1, and works for applications and applets. This means that any Java code that currently uses traditional Swing menus can switch to using the fisheye menus with a one-word change by replacing "`new JMenu()`" with "`new FishEyeMenu()`"[1].

The standard approach to implementing fisheye distortion techniques is to compute a "Degree of Interest" (DOI) function for each element to be displayed. The DOI function calculates whether to display an item or not, and it calculates the item's size. Typical degree of interest functions include both the distance of an item from the focus point as well as the item's a priori importance [7]. Thus, certain landmark items may be shown at a large size even though they are far from the focus point.

The fisheye menu uses a very simple DOI function that only includes distance from the focus point, and does not use a priori importance. A simple function that captures the essence of the fisheye menu is shown in Figure 5. It keeps several menu items near the focus point at the maximum size, where the exact number is specifiable. Then, the menu items get smaller, one point in font size at a time until the minimum font size is reached at which time, all more distant items stay at the minimum font size.

Using this DOI function, the fisheye menu calculates the largest minimum size font that will result in a menu that fits on the screen. If there are so many items in the menu, or if there is so little available screen space that there is not enough room for the menu, then the DOI function parameters are adjusted so there is enough room. First, the focus length is reduced. If there is still not enough room when the focus length is set to 1, then the maximum font size is reduced.

Complexities

In practice, the DOI function is actually a little more complex than just described for two reasons. The first reason is that we want the menu items to be visually stable outside of the focus area. That is, if the focus is on the first half of the menu, it is important that the second half of the menu doesn't move at all as the focus changes. The fisheye menu is stable using the above DOI function when the focus is not near one of the ends of the menu. However, when it is near the ends of the menu, there is a surprising side effect of the algorithm, which results in the entire menu shifting.

Since we render each item based on the position of the item before it, one item alone changing size will slide all other lower menu items up or down. Moving the focus in the middle of the menu doesn't cause a problem because for every item that gets bigger, another items gets smaller by the same amount. To understand the issue here, let us look at the simplest case where the focus is on the first item in the menu. In this case, there are no items before the focus item to get rendered, and the items after the focus item get smaller until the minimum size is reached. Compare this with the focus being on the second item in the menu. Now, one item before the focus is rendered at a large size while the items after the focus get smaller in the same way. Thus, more space is taken altogether, and the entire menu shifts down a little bit. The entire menu continues to grow as the focus moves down from the end until the distortion no longer goes to the end of the menu and the menu becomes stable.

Our solution is to increase the size of the focus area just enough to account for the smaller number of focus items when the focus point is near the menu end. This way, the total amount of space used by the focus area is always constant, and the entire menu remains visually stable.

The fisheye menu uses this modified DOI function to calculate the required size of the popup menu. This leads to the second reason that our DOI function is more complex in practice. We use integer calculations since text is only rendered in integer sizes, and so the popup menu size can end up being substantially smaller than the available space. We want to use as large a menu size as possible since the bigger the menu is, the more items we can render in a large enough font to read, and the more usable the fisheye menu will be.

Once the minimum size font is calculated, a menu that uses all the available screen space is created. Then the DOI function is modified using the same technique that we used to solve the first problem - the focus area is expanded until the text fills up the full menu space.

One remaining issue has to do with the alphabetic index. Since the index characters are always rendered at full size, they would overlap each other when they are far from the

[1] Note that the online applet uses Java 2 to decrease the portability problems associated with accessing Swing from Java 1.

DOCKET ALARM

Explore Litigation Insights

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

Real-Time Litigation Alerts

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

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

Advanced Docket Research

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

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

Analytics At Your Fingertips

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

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

API

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

LAW FIRMS

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

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

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.