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 [20]. 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, 11, 21]. Several variations of the fisheye technique have been explored from graphical maps [16] to space-scale diagrams [8] to 3D [15] and 2D tables [13]. Some applications of fisheye distortion techniques have been carefully evaluated, often finding a significant advantage to fisheye views [5, 10, 17].

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

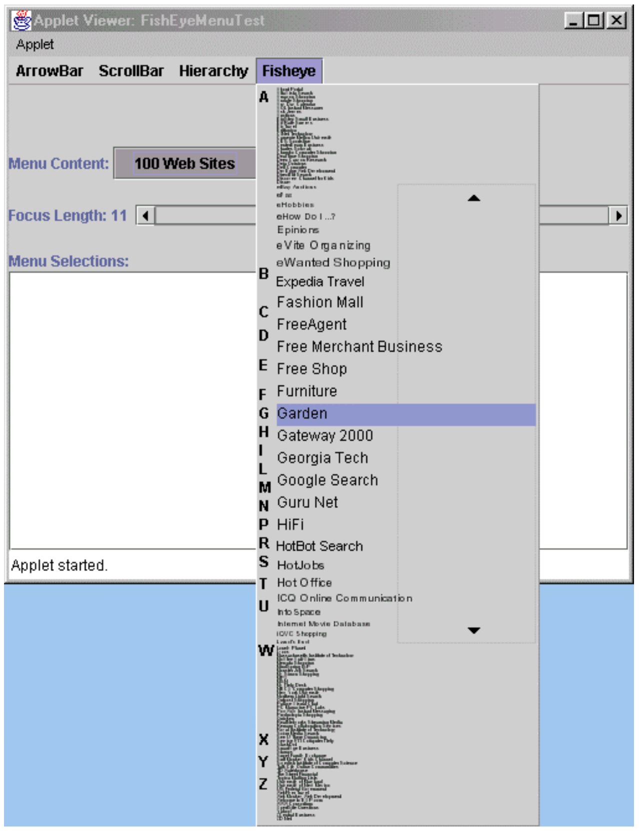

Figure 1: A screen shot of the fisheye menu in use. This shows 100 web sites taken from the most popular list of PC Magazine.

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.

Selecting items from menus is another well-studied area, and the trade-offs of menu design are well understood [9, 12]. 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 for more than about 35 elements, 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 [14], 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 a user a time proportional to the menu length to move a pointer to an item on the menu (on average) [6, 18]. 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. 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 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 very 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, thus bringing

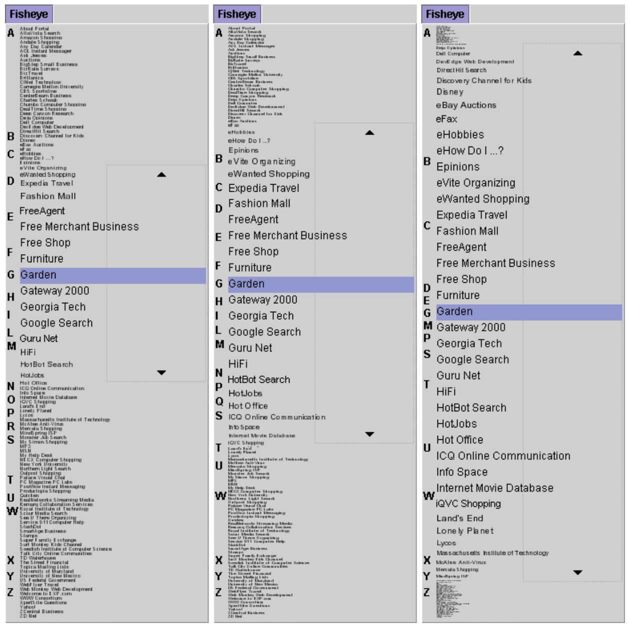

Figure 2: The same menu of 100 items displayed with varying focus lengths (7, 12, and 20). There is a fixed maximum font size.

that portion of the menu into focus at which point they can read the menu items and select the particular item 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

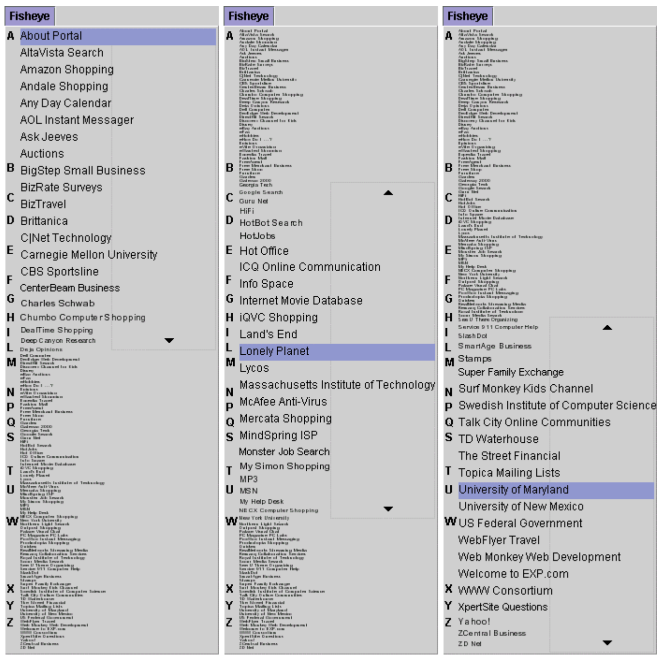

Figure 3: The same menu displayed with the cursor at three positions.

fisheye focus changed, the index letters would move around, following the items. This turned out to be not only distracting, but also 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. In fact, Fitt's law shows us that the time to select an item is inversely proportional to the target item's size. For example, if a fisheye menu item is effectively 3 pixels high compared to a traditional 18 pixel high item (12 pixel font and 6 pixel space), it will take 6 times longer to select the item.

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 Fitts' law 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].

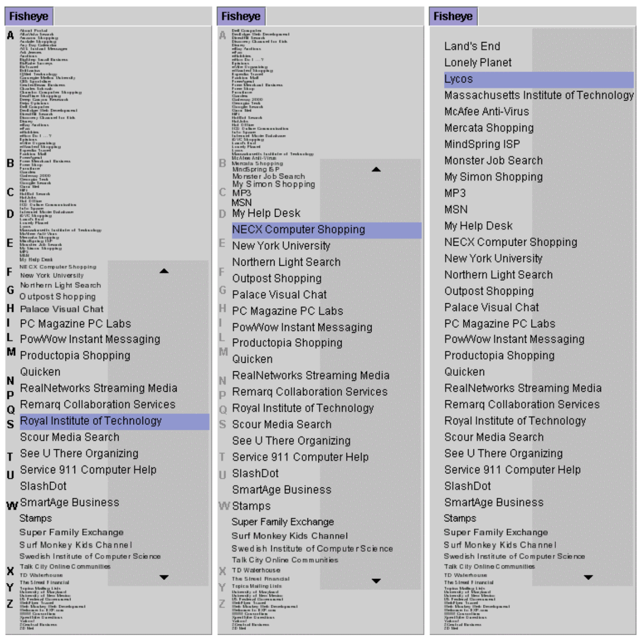

Figure 4: A fisheye menu in focus lock mode whose focus area is being extended upwards

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, the focus area is extended, and the arrows move accordingly. The users can thus discover that the focus can 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 pixel in font size at a time until the minimum font size is reached at which point, 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. Thus, the fisheye menu always does the best it can in presenting a large number of large items in the menu, given the constraints of available space, following the hints of the suggested maximum font and focus length parameters.

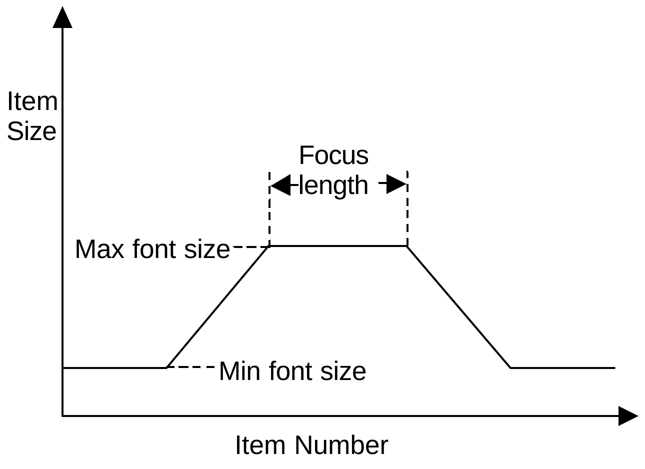

Figure 5: The basic Degree of Interest function used for the fisheye menu.

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 the entire rest of the menu 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.

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

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 focus area, since the associated menu items at that point are quite small. The fisheye menu avoids this overlapping problem by simply not rendering indices that would overlap with another. Thus, in the periphery, not every index character is shown.

The fisheye menu is implemented by pre-calculating the size of every item and the space between each item for each focus position, and storing that information in look-up-tables. This pre-calculation is necessary in order to calculate the position of the index letters. This also improves performance since there is very little calculation during rendering. One final, but important optimization is the use of region management. Since the fisheye menu is visually stable, only the changing focus portion of the menu changes as the pointer moves. Our implementation keeps track of the area on the screen that changes, and only renders that portion. Thus, for a menu of 200 items, typically less than 30 items need to be rendered for each mouse movement.

EVALUATION

We conducted a pilot study of fisheye menus comparing user preference of them against the three menu mechanisms commonly used today: arrow buttons to scroll up and down, scrollbars, and hierarchies. The intent of this study was to get a preliminary idea of whether fisheye menus had potential. We did not expect that the results of this study would provide a definitive understanding of whether fisheye menus were faster, more appropriate, or preferable for tasks. Rather, we hoped to get a rough idea of user's preferences that would let us know if our intuitions were realistic, and to inform future evaluations.

We picked 10 users that were not from our lab, and were not familiar with fisheye menus before the study. Five of the subjects were computer science students with programming experience, and five of the subjects were administrative staff that work in our building, and did not have programming experience. We felt that looking at programmers vs. non-programmers was important because fisheye menus are somewhat technical, and we sensed that people with less technical experience may not feel immediately comfortable with them. As it turned out, there was a difference between these two classes of users which will be reported in the *Results* section.

Seven of the subjects were female and three were male. Five were in there 20's, two were in their 30's, two were in their 40's, and one was over 50. All but one reported using computers more than 20 hours per week.

The test was entirely automated using a custom Java program. The program requested demographic information, and explained that the purpose of the test was to get feedback on the four types of menus for selecting an item from a list. The subjects were then instructed to try out each of the menu types, spending as much time as they liked. At that point, they were instructed to ask any questions about how the menus worked (the test was administered by the author of this paper.)

The four menu types were labeled ArrowBar, ScrollBar, Hierarchy, and Fisheye. All menu items were ordered alphabetically. The ArrowBar was implemented with arrows at the top and bottom of the screen. When the arrows were pressed, the list would scroll at a rate of 20 items per second. The ScrollBar was implemented with a standard scrollbar on the right side of the menu that could be used to scroll the menu. The Hierarchy was constructed with one menu item for each letter of the alphabet. Menu items were placed in cascading menus under the first letter of the text of that item. Finally, the Fisheye menu was that described in this paper. Each of these menus are available for trial at the fisheye menu website.

Then, the subject was instructed to select three different specific items from each menu. Each menu was populated with 100 websites that were selected from the list of most popular websites from PC magazine (plus, a few universities were added.) The items that the subjects were told to select were chosen from near the beginning, middle, and end of each list. The subjects were also asked to browse the lists for a website they would like to visit. No feedback was given, nor was information logged as whether to the subjects correctly selected the specified item.

The subjects were asked to rate the menus. They were asked to rate each menu using a 9 point Likert scale according to seven characteristics taken from QUIS – the Questionnaire for User Interface Satisfaction [19]. The seven characteristics were:

- terrible – wonderful
- frustrating – satisfying
- difficult – easy
- slow – fast
- hard to learn – easy to learn
- boring – fun
- annoying – pleasant

Finally, the subjects were asked to rank the four menu types in order of preference for goal-directed tasks and browsing tasks. They were also offered the option of typing in any comments they had about the four menu types.

Results

The average subjective satisfaction of the four menu types was recorded for all users, and separated by programmer vs. non-programmer. For all users, on a scale from 1 – 9 (with 9 being most positive), Hierarchy was the favorite (6.8), Fisheye (6.4) was rated slightly higher than Scrollbar (6.2), and ArrowBar (4.9) was the lowest.

When split by programmer, an interesting difference appears. The ratings of ArrowBar and ScrollBar did not change very much, but Fisheye and Hierarchy did. For programmers, Fisheye (7.0) and Hierarchy (6.9) were about the same. For non-programmers, the spread between Fisheye (5.8) and Hierarchy (6.8) substantially increased.

When looking at the individual questions, we see that the subjects had widely differing opinions about Hierarchy vs. Fisheye in different categories. Hierarchy was preferred over Fisheye in the three categories of 'frustrating – satisfying', 'hard – easy', and 'hard-to-learn – easy-to-learn'. However, Fisheye was preferred over Hierarchy in the four categories of 'terrible – wonderful', 'slow – fast', 'boring – fun', and 'annoying – pleasant'.

When asked to directly rank the four menu types in order of preference, there was a difference for goal-directed and browsing tasks (Figure 6). For goal-directed tasks, ArrowBar and ScrollBar were clear losers with Hierarchy just beating out Fisheye. For browsing tasks, ArrowBar was at the bottom, ScrollBar and Hierarchy were about tied in the middle, and Fisheye was the most preferred. However, the large standard deviation of Fisheye shows that there was a broader range of reaction. Some users ranked it about the same as ScrollBar and Hierarchy, and some users ranked it much higher.

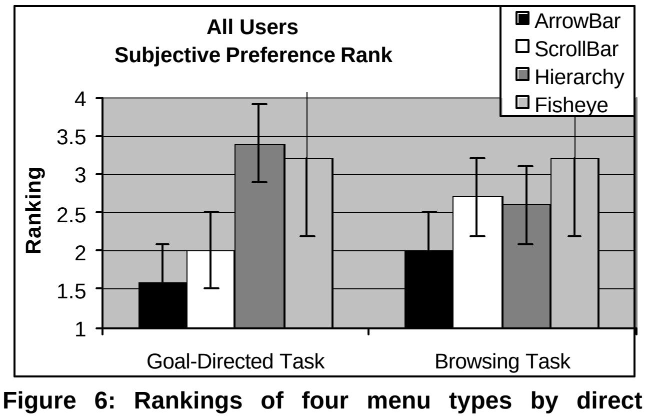

Figure 6: Rankings of four menu types by direct comparison for goal-directed and browsing tasks. Error bars mark 1 standard deviation.

When separated out by programmer vs. non-programmer, there was a similar effect as with the satisfaction ratings. Programmers preferred Fisheye to Hierarchy in all cases, with a small margin (0.2) for goal-directed tasks, and a big margin (1.0) for browsing tasks. Non-programmers preferred Hierarchy to Fisheye for goal-directed tasks by a margin of 0.6 and they were tied for browsing tasks.

The subjects' comments were informative and mirrored the rating and ranking results. Two non-programmers specifically said that they did not like fisheye at all. The other eight subjects all liked fisheye, and frequently had concerns about the difficulty of learning to use it. However, they also expressed optimism that with more training, it would become more enjoyable and perhaps preferable. A few typical comments were:

> *"Fisheye was the most difficult to learn yet with continued use may actually become the most useful."*
>
> *"ArrowBar and ScrollBar are boring but very easy to use. I am used to it. Hierarchy and Fisheye are very interesting."*
>
> *"Once one understands that one has to go to the colored area in Fisheye it becomes easier. But if one doesn't know that it's frustrating."*

Analysis

We learned several things by conducting this preliminary study of fisheye menus. While the study contained a small number of subjects and the results were not analyzed statistically, there are some trends. The test was administered without a description of what fisheye menus were or how they worked. Instead, the subjects were told to play with them for as long as they wanted and only then could they ask questions.

By observing this initial exposure to fisheye menus, and by responding to the subjects' questions, it was clear that at least in the minute or two that they tried them, most subjects did not understand how to use the fisheye menu fully. All of the subjects quickly discovered that moving the mouse up and down on the left side of the menu operated the basic fisheye functionality. However, several were confused about the exact function of the alphabetic index on the left side. Several users tried clicking on them – which just selected the item that was currently highlighted. After one or two tries with this, they then realized that the index was just informative, and not interactive.

A more important problem was that only a single subject truly discovered how the "focus lock" mode on the right side of the menu worked. Despite the visual feedback, subjects were just not expecting to have different behavior when the mouse pointer was on different sides of the menu. Some subjects never moved the pointer to the right side and so never discovered that behavior at all. Other subjects moved the pointer to the right side of the menu accidentally or erratically. They just noticed that the menu would sometimes change behavior in an inconsistent manner. They did not correlate the change in menu behavior with the side of the menu that the pointer was over.

Once the subjects were done exploring the menus and asked questions, the focus-lock mode was explained. Interestingly enough, all 10 subjects completely understood how it worked in just a few seconds of explanation. Thus, the visual design of the menu clearly needs some work to make the focus-lock mode more discoverable.

Another major lesson learned from these studies is that subjects' response varied widely. Looking at the average results only tells part of the story. Two of the subjects did not like the fisheye menus at all. It had nothing to do with the difficulty they had to discover how they worked. Rather they just didn't like them. One of those users reported that the small menu items made her feel badly because she felt that her eye sight was poor.

On the other hand, several of the users were eager to start using fisheye menus in their regular work immediately. This bimodal preference indicates that fisheye menus, if deployed in an application, should always be optional. Some users are likely to prefer them, and some are likely not to.

The last lesson we learned from this study is that application designers should consider the use of scrollbar and hierarchical menus instead of the traditional arrow menus used by default by current operating systems. Or better yet, let users set an option to specify how long menus will be presented.

The ArrowBar menu was the clear loser in all cases. Subjects felt it was boring, slow, and frustrating. Yet, this is the most common type of long menu in commercial systems. The ScrollBar menu, on the other hand, provided a nice compromise for goal-directed and browsing tasks, and was generally enjoyed by users. While the Hierarchy menu was often preferred for goal-directed tasks, the same menu will be used in different ways by different users. Some users will know exactly what they want while some will browse. So, the Hierarchy menu should be used cautiously if at all, and only when it is clear that users know exactly what they are looking for.

Expert Timing

We also performed a very simple test to see how fast an expert could use each of the menu types. The author of this paper selected an item from the middle of the menu from each of the menus 10 times working as quickly as possible. The fastest time was recorded. This was done for the 100 web sites, and also for a list of 266 countries.

For the 100 websites, the times were: ArrowBar (3.4 secs); ScrollBar (2.2 secs); Hierarchy (1.5 secs); Fisheye (1.7 secs). For the 266 countries, the times were: ArrowBar (8.8 secs); ScrollBar (2.6 secs); Hierarchy (2.1 secs); Fisheye (2.3 secs).

These timing results match closely with the subjective preferences for goal-directed tasks, and so suggest that these data may reflect a broader trend than would be indicated by so few subjects.

CONCLUSION

Selecting an item from a list is an important and frequent task. We have presented here fisheye menus, a new mechanism that supports this kind of selection. Based on our preliminary evaluation, we believe that this approach is promising. It clearly is not for all users, but just as clearly, it is preferred by many users, so at this point we recommend considering fisheye menus for optional use where selection from a long list is required.

We plan on continuing the investigation of fisheye menus by conducting a controlled empirical evaluation, including analysis of the speed users can select items with the different menu types. We also will consider other menu types such as matrix or multi-column layouts, and will look at other factors such as the number of items in the menu.

ACKNOWLEDGEMENTS

I appreciate the thoughtful comments of the members of the Human-Computer Interaction Lab who helped me fine-tune the details of the fisheye menus. In particular, I thank Harry Hochheiser who suggested the alphabetic index, and the subjects who volunteered their time to give me valuable feedback.

REFERENCES

1. Abrams, D., Baecker, R., & Chignell, M. (1998). Information Archiving With Bookmarks: Personal Web Space Construction and Organization. *In Proceedings of Human Factors in Computing Systems (CHI 98)* ACM Press, pp. 41-48.
2. Ahlberg, C., & Shneiderman, B. (1994). The AlphaSlider: A Compact and Rapid Selector. *In Proceedings of Human Factors in Computing Systems (CHI 94)* ACM Press, pp. 365-371.
3. Beck, D., & Elkerton, J. (1989). Development and Evaluation of Direct Manipulation Lists. *SIGCHI Bulletin, 20*(3), pp. 72-78.
4. Dill, J., Bartram, L., Ho, A., & Henigman, F. (1994). A Continuously Variable Zoom for Navigating Large Hierarchical Networks. *In Proceedings of IEEE International Conference on Systems, Man and Cybernetics* IEEE, pp. 386-390.
5. Donskoy, M., & Kaptelinin, V. (1997). Window Navigation With and Without Animation: A Comparison of Scroll Bars, Zoom, and Fisheye View. *In Proceedings of Extended Abstracts of Human Factors in Computing Systems (CHI 97)* ACM Press, pp. 279-280.
6. Fitts, P. M. (1954). The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology, 47*, pp. 381-391.
7. Furnas, G. W. (1986). Generalized Fisheye Views. *In Proceedings of Human Factors in Computing Systems (CHI 86)* ACM Press, pp. 16-23.

8. Furnas, G. W., & Bederson, B. B. (1995). Space-Scale Diagrams: Understanding Multiscale Interfaces. *In Proceedings of Human Factors in Computing Systems (CHI 95)* ACM Press, pp. 234-241.

9. Hochheiser, H., & Shneiderman, B. (2000). Performance Benefits of Simultaneous Over Sequential Menus As Task Complexity Increases. *International Journal of Human-Computer Interaction,* (in press).

10. Hollands, J. G., Carey, T. T., Matthews, M. L., & McCann, C. A. (1989). Presenting a Graphical Network: A Comparison of Performance Using Fisheye and Scrolling Views. *(Third International Conference on Human-Computer Interaction)* Elsevier Science Publishers, pp. 313-320.

11. Mitta, D., & Gunning, D. (1993). Simplifying Graphics-Based Data: Applying the Fisheye Lens Viewing Strategy. *Behaviour & Information Technology, 12*(1), pp. 1-16.

12. Norman, K. (1991). *The Psychology of Menu Selection: Designing Cognitive Control at the Human/Computer Interface*. Ablex Publishing Corp.

13. Rao, R., & Card, S. K. (1994). The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information. *In Proceedings of Human Factors in Computing Systems (CHI 94)* ACM Press, pp. 318-322.

14. Robertson, G., Czerwinski, M., Larson, K., Robbins, D. C., Thiel, D., & van Dantzich, M. (1998). Data Mountain: Using Spatial Memory for Document Management. *In Proceedings of User Interface and Software Technology (UIST 98)* ACM Press, pp. 153-162.

15. Robertson, G. G., & Mackinlay, J. D. (1993). The Document Lens. *In Proceedings of User Interface and Software Technology (UIST 93)* ACM Press, pp. 101-108.

16. Sarkar, M., & Brown, M. H. (1992). Graphical Fisheye Views of Graphs. *In Proceedings of Human Factors in Computing Systems (CHI 92)* ACM Press, pp. 83-91.

17. Schaffer, D., Zuo, Z., Bartram, L., Dill, J., Dubs, S., Greenberg, S., & Roseman, M. (1997). Comparing Fisheye and Full-Zoom Techniques for Navigation of Hierarchically Clustered Networks. *In Proceedings of Graphics Interface (GI 97)* Canadian Information Processing Society, pp. 87-96.

18. Sears, A., & Shneiderman, B. (1994). Split Menus: Effectively Using Selection Frequency to Organize Menus. *ACM Transactions on Computer-Human Interaction, 1*(1), pp. 27-51.

19. Slaughter, L. A., Harper, B. D., & Norman, K. L. (1994). Assessing the Equivalence of Paper and On-Line Versions of the QUIS 5.5. *In Proceedings of 2nd Annual Mid-Atlantic Human Factors Conference* pp. 87-91.

20. Spence, R., & Apperley, M. (1982). Data Base Navigation: an Office Environment for the Professional. *Behaviour & Information Technology, 1*(1), pp. 43-54.

21. Spenke, M., Beilken, C., & Berlage, T. (1996). FOCUS: The Interactive Table for Product Comparison and Selection. *In Proceedings of User Interface and Software Technology (UIST 96)* ACM Press, pp. 41-50.