Pies: Implementation, Evaluation, and Application of Circular Menus

Don Hopkins
Jack Callahan
Mark Weiser

Heterogeneous Systems Lab
Computer Science Department
University of Maryland, College Park

ABSTRACT

Menus on computer displays are usually a linear row or column of choices. We propose an alternative to linear menus, called the Pie menu. The choices of a Pie menu are positioned in a circle around the menu center. The direction in which the cursor is moved makes the menu selection, and the length of motion is available as a second input. Pie menus encompass most uses of linear menus, while introducing many more. In this paper, we describe our implementation of Pie menus in the MIT X window system and in the SunView window system. We also evaluate Pie menus vs. linear menus in a controlled experiment, where we find that computer-naive subjects made selections faster, and with fewer mistakes, using Pie menus. Finally, we speculate on the theoretical advantages of using Pie menus in new applications (or re-organized old applications). Three examples of such advantages are: (1) the return of two kinds of information from every Pie selection (direction and distance), (2) selection precision increases with distance, and (3) the existence of natural opposites (left/right, up/down).

NOTE: This paper was accepted in 10/88 by CACM (with minor revisions) BUT IT WAS NEVER REVISED - and therefore NOT PUBLISHED

Nevertheless printed copies of this paper were circulated, so it is cited in patents and other papers.

For this reason we later on scanned the submitted version and posted it online.
1. Introduction

We are proposing a new menu format, which we call Pies menus. We believe Pies menus offer significant advantages over other kinds of menus, and are different enough that to do them justice requires examination from many viewpoints. This paper offers three viewpoints: the implementation of Pies in two existing window systems (X and SunView), the evaluation of Pies vs. traditional menus in a controlled experiment, and the potential for Pies over traditional menus for future applications. We believe the three viewpoints converge to an implementable, efficacious, and exciting idea. Before starting into the viewpoints, the introduction below reviews traditional menus, and introduces Pies.

1.1. Traditional Menus

When it comes to presenting a list of choices to the user, most computer systems have been limited, largely by the available hardware and software, to a linear format (Figure 1). As a concept, linear menus have the following properties (adapted from Shneiderman[Shneiderman

<table>
<thead>
<tr>
<th>WindowOps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
</tr>
<tr>
<td>Logout</td>
</tr>
<tr>
<td>Open</td>
</tr>
<tr>
<td>Close</td>
</tr>
<tr>
<td>Hardcopy</td>
</tr>
<tr>
<td>Shrink</td>
</tr>
<tr>
<td>Shape</td>
</tr>
<tr>
<td>Move</td>
</tr>
</tbody>
</table>

Figure 1: A linear menu.
(1) The selection made from a linear menu is based upon the absolute cursor location within a square coordinate grid within the menu. Usually this grid is one-dimensional (i.e. a list of items) but occasionally it is two-dimensional (and array of items.)

(2) The top of a linear menu is usually a dead zone, where no item is selected.

(3) If the cursor is outside the menu then (usually) no item is selected.

(4) Menus are activated from mouse actions in two formats: pull-down (menu appears at a fixed label on screen) or pop-up (menu appears anywhere within a fixed area, occasionally the whole screen)

Even so, many systems have used the two dimensional nature of the computer display to the advantage of certain menu applications. Many flight simulation program, for example, layout directional headings in a typical compass format (Figure 2).

Item placement in menus has been an important research topic for many years. Menu organization is typically divided into three types[Dray et al 81]: alphabetical/numerical,
categorical (functional), and random ordering. It is generally agreed that the performance of subjects (i.e. time to seek a target) with different placement styles converges with practice[Card 82, Perlman 84]. Further studies [McDonald et al 83, Perlman 84] revealed that a functional placement of items is superior when the task domain is unambiguous to the user whereas an alphabetic organization can be useful in uncertain task descriptions. Schultz and Curran[Schultz and Curran 86] offer more evidence that broader, shallower menus are better. All of these studies have concentrated on the linear display format.

Most commercial window systems available today make a general package of linear menu presentation available to the programmer. Libraries of software of menu display routines are widely used as a default by programmers of many window systems and applications[Xerox 85, Sun 86]. These libraries often have default display mechanisms which could easily accommodate other presentation methods without changing a single line of application code. Pie menus are such an alternative presentation method, one with many advantages and few disadvantages (see section 4).

1.2. Pie Menus

The Pie menu is a concept which can have many implementations, two of which are described in a later section. As a concept, Pie menus have the following properties:

1) The selection made from a Pie menu is based upon the angle between the position of the mouse down event (where the menu is centered) and that of a later mouse up event. Sometimes, in some implementations, there may be intervening mouse clicks before the final selecting event.

2) The center of a Pie menu contains a dead zone a few pixels wide. When the cursor is in the dead zone no selection is made. Normally when a Pie menu first appears the cursor
is warped\textsuperscript{3} into the dead zone.

(3) Pie menus usually permit selection even when the cursor is outside menu borders.

(4) Pie menus optionally return the distance from the menu center, as well as the item selected.

Conceptually Pie menus can parallel linear menus in functionality. Pie menus can be nested, either 'layered' or 'walking' (Figure 3). Pies can be thought of as simply an alternative user interface into a common underlying selection process.

Notice that Pies have two important differences from normal menus on simple selection tasks: all items are near to and equidistant from the initial mouse position, and greater resolution of selection is easily available via greater motion. The last point would have significance in a situation made noisy by any of the following factors: human (hand jitter), electromecahnical (mouse jitter), environmental (room jitter). By moving the mouse a larger distance from the Pie menu center a more accurate selection is assured. This feature of Pie menus also facilitates 'mouse-ahead'. Our experience has been that even an experienced and confident menu user will not mouse ahead to a linear menu item more than two or three from the top, while an experienced Pie menu user (one of us\textsuperscript{4}) easily mouses ahead to any item on an eight item Pie menu.

Pies can easily be used as "pull-down", as well as "pop-up", menus. To keep the menu below the selection only a partition of a Pie can be used, resulting in a fan-like menu display. Figure 4 shows a pull-down Pie menu.

\textsuperscript{3}Warping refers to moving the on-screen cursor when the user has not moved the mouse. Proper warping is essential to good menu "feel" when popping up menus near the edge of the screen. See additional discussion in section 2.1.

\textsuperscript{4}Hopkins.
Figure 3: Nested pies, both ‘stack-of-cards’ and walking.
2. Pie Implementation

2.1. Introduction and problems

We do not know of Pie menus in any commercial systems. Imaginative menu formats, however, are inevitable with the latest advances in window management systems[Gosling 85]. Window imaging systems using technology from laser printing protocol standards such as Interpress[Xerox 84] and PostScript[Adobe 85] will make it possible to display arbitrarily shaped windows effectively on a bitmapped display. Circular, elliptical, and other window shapes (floral?) menus will be possible to implement easily\(^5\). There are some obvious advantages to this organization for particular applications: compass directions, time, angular degrees, and spatially related names are some design ideas that work well with Pie menus. However, as the next section shows, Pies win over linear menus even for ordinary tasks.

---

\(^5\)Many of the more fanciful figures in this paper were generated directly from Postscript programs.
We have implemented Pies in two different window systems: X and Sunview. The next subsection discusses details specific to these implementations. The remainder of this subsection discusses issues that any implementation will confront: layout within windows, and menu warping near screen edges.

There are two kinds of layout problems. The first is that, for small numbers of items, Pies tend to take up more space than linear menus because of the non-uniform space in a Pie selection ‘slice’. A triangle is hard to fill completely with most text or graphics found in menu items. Both of our implementations used strictly horizontal or vertical letter placement with Pies, but even slanted letters would not fit much better. One could argue that the unoccupied space in Pie slices is not wasted, since it participates in selection, but if screen space is critical it is still annoying.

The second layout problem is simply how to arrange text around a Pie menu. Which way should letters travel, or if images are used, where should they be within the circle? Should the radius of the Pie enlarge to make all text and/or pictures fit naturally?

Both of our implementations position the midpoints of each text item equally around the Pie. For a Pie with $m$ items, the $k^{th}$ item will be positioned with its upper-left coordinates as follows:

$$x = \frac{(d \sin(\theta)+d-w)}{2}$$
$$y = \frac{(d \cos(\theta)+d-h)}{2}$$

where $d$ is the diameter of the Pie menu bounding box, $\theta = 2\pi \frac{k}{m}$ radians, $w$ and $h$ are the width and height of the $k^{th}$ item, and coordinate $(0,0)$ is the upper-left corner of the Pie box. Using these formulas, the values of $x$ and $y$ must still be limited so that the menu item completely fits in the Pie box, as follows:
\[ z = \max(0.0, \min(d - w + 2, \max(2.0, x))) \]
\[ y = \max(0.0, \min(d - h - 2, \max(2.0, y))) \]

The values of 2.0 keep the Pie items at least 2 pixels away from the box. The Sunview implementation can use rectangular images called 'Pixrects' as well as text, and these are positioned exactly as for text.

The Sunview implementation attempts to automatically pick a diameter for the Pie box. It uses the heuristic \( d = 3MZ + 1 \), where \( M \) is the largest dimension, either width or height, of any item, and \( Z \) is the size of the 'dead zone' at the center of a Pie where nothing is selected (typically 5 pixels wide). This heuristic works well if the items are approximately equal sized, but looks crowded if there is wide variation in size or there are many items. An alternative, not implemented, would be to actually compute the space taken by each item, and enlarge the Pie until a minimum inter-item spacing is achieved. The computational cost would not be very high, and would be incurred only at the time of menu specification.

One of our early concerns was for what we thought would be the high cost of computing selections from mouse positions. While a linear menu can simply compare coordinates, a Pie needs trigonometry. Our first implementation, in X, goes to great length to avoid calling \texttt{arctan}. In retrospect this was unnecessary, since \texttt{arctan2} on our hardware is fast enough (2.3 milliseconds on a Sun-3 with 68881 FP chip, 6.6 milliseconds on a Sun-2 with software-only FP) that it does not interfere with accurate tracking. A menu is only tracking selections for a few seconds at a time, and an overhead of 6.6 milliseconds per cursor update still leaves plenty of idle cpu time.

The menu warping problem is common to all popup menu systems, but is particularly acute for Pie menus because of the requirement that the cursor start in the center of the menu. Normally a menu will popup (or pulldown) starting at the position of the cursor when the menu is first activated. However, if the cursor is close to the edge of the screen the menu may
not completely fit, and must be displayed in a slightly different screen position. Because of mouse-ahead and other user expectations, if the menu is displayed askew then the cursor must also be moved so that it is in the same position relative to the menu that it would have been had the menu not moved. This automatic moving of cursor and menu near the edge of the screen is called 'mouse warping'.

Linear menus are normally smaller than Pie menus, and the cursor normally starts near the edge of a linear menu, often at the top edge or the top item. The large size of Pie menus, and the need to move the mouse to the middle of this larger size, make good warping more important for Pies than for linear menus.

The most basic warp algorithm, used in our Sunview implementation, is to simply force the menu onto the screen, and then warp the mouse to the middle. Unfortunately, this makes mousing ahead impossible, since all mouse motion before the warp is lost. A better algorithm, which we used for a time in X, looks at the offset between the mouse down event and the current mouse location, and warped the mouse to the center of the (repositioned) menu plus that offset. However, this complicates things if there are more input events pending. The positions associated with pending events are still relative to the unwarped mouse position, and in particular the angle towards the pending mouse up event that specifies the current selection is different after the mouse has been warped. Also, if the last pending event is a mouse down, the angle between that event and the current mouse position changes.

Our most sophisticated solution is in our current X implementation. There, we only put up a menu when there are no pending mouse button events. When there are events pending, they are acted upon without putting up a menu. When there are not, and more are needed to complete a selection, a menu is actually put up. If the menu would not fit completely on the screen, it is moved by the appropriate offset back onto the screen, and the mouse is warped to
its current location plus that same offset. Normal tracking then begins.

For instance, if at the time the mouse down event is received, the mouse up event that completes the selection is already pending, then there is already enough information to make the selection. It is not appropriate to look at the current mouse position, because the selection has already been completely specified. At this point, putting the menu up serves no purpose but feedback, and slows down interaction. The selection can be made and acted upon without even showing a menu, if feedback is not required, or if acting on a selection provides the feedback.

2.2. Implementation Details

Our implementation of Pie menus in the Ultrix window manager uwm of the X Window System (V10r4)[Gettys 86] uses no new window management techniques, and in fact places the circularly arranged selection items within a rectangular popup window (Figure 5). Adding Pie menus to uwm required writing 500 new lines of code, most of them changes to a single module. The task took one of us (Don Hopkins) about a month of part-time work, some of which was experimenting with display algorithms.

At the time Pie menus were first implemented (June 1986) there was not a standard menu package in X. Different applications used their own libraries. Therefore, our implementation was put into one of those applications, 'uwm'. Uwm is the most flexible of the X window managers, and is extensively customized for individual users by a special startup file ('.uwmrc'). Customization can include the specification of new popup menus, and the action to be taken when menu items are selected.

We augmented the language of the .uwmrc file to include new menu specification keywords. There is a global keyword, 'piemenus', which when encountered causes menus to default to Pie menus. The new keyword 'Pulldownmenus' has the opposite effect. There are also new keywords for individual menu specifications: 'piemenu' and 'pulldownmenu'.
'Pulldownmenu' forces the usual uwm menu, 'piemenu' the new style, regardless of the global keyword. For Pie menus, the user can also specify the direction and distance of the first item of the menu. By default the first item is put at the '12 o'clock' position, at a distance determined by the 'radius' global keyword.

The Pie code was integrated into the existing uwm menu code so that no part of uwm except the menu implementation (and the .uwmrc parser) knew anything about Pies. The result was a very smooth integration, upward compatible with existing use but, by setting one variable, upgradable to make all menus Pies.

There is now (March 1987) a second implementation of Pie menus, mostly compatible with the normal Sun Microsystems menu library, and living in SunView (see Figure 6). This required changing no lines of existing Sun source code, but is itself about 500 new lines of
code. Existing modules can use Pie menus by compiling against the new library. Not all features of the Sun menu library have been implemented, just all those that are used by the background menu process, 'suntools'. This implementation took one of us (Mark Weiser) a (snowed-in) weekend. It supports the following features of Sunview menus: nested (walking) menus, images or text for items, and user-procedure callback on selection. It does not support generating menus on-the-fly, or every different Sunview method of menu specification (of which there are many). It differs from our X implementation in having automatic diameter
calculation, optional image-based items, and less sophisticated warping.

3. Evaluation

3.1. Hypotheses

We performed a controlled experiment to test two hypotheses: that Pie menus decrease the seek time for menu items and that Pie menus are especially useful in menu applications suited for a circular format or diametrically opposed item sets. Only a summary of the experiment is given here—full details will appear elsewhere[Callahan et al '87].

We expected Pie menus to do better primarily because of the decreased distance to the target in Pies when compared to linear menus. The distance to an item in any menu style can be defined as the minimum distance needed to highlight the item as selected. In both menu styles, this is defined by a region rather than a point. This region is typically of greater area than the actual target (Figure 7). Once the cursor has entered the region, the item is highlighted as feedback to the user. Note that the regions may extend beyond the menu window itself. Even with linear menus where the cursor may be placed initially in the middle of the list, the outer item choices suffer from their distance from the initial cursor location. Pie menus, in fact, enjoy a two fold advantage because of their unique design: all items are equidistant, and that distance is in fact quite small when measured as the cursor motion needed to select an item.

3.2. Experimental Design

The experiment is fairly straightforward and is constructed in a 2x3 randomized block design [Kirk68] (Table 1).
Each cell is an element of the cross product of menu type and task. A typical Pie task would be a compass because it seems best suited functionally for Pie menus. List of elements, like OPEN/CLOSE and UP/DOWN, whose meanings are antonyms are also classified as Pie tasks. Lists, like numbers, letters and ordinals, are best suited for linear menus and are thus classified as linear tasks. Groups of menu items that have no relation to each other fall in the unclassified category. As it turned out, Pies were better for all three groups of menu items.
There are a total of 15 menus, a group of 5 for each task type (column). Subjects performed the experiment for all cells in the experiment matrix in random order in accordance with a randomized block design [Kirk68]. The subjects saw the same menu four times, twice in each menu format. Each subject therefore saw a total of 60 menus. The target item in each menu varied for the menu type and were balanced in their placement.

Subjects were taken from the University of Maryland Psychology Department Subject Pool. 26 subjects performed the experiment in addition to 16 pilot subjects. All subjects were undergraduate students with little or no mouse experience. They were rewarded with 1 extra credit point for participating.

3.3. Materials

The experiment was run on a Sun Microsystems Workstation Model 3/160 running MITs X window system with the uwm window manager modified for Pie menus. (This modification is described earlier in this paper). To make a selection, the user invokes the menu by pressing a mouse button, continues to hold the mouse button down and moves to an item which is then highlighted, and releases the mouse button confirming the selection.

A typical sequence of events for the user is shown in Figure 8. The computer posts the target name at the top of the screen, the user invokes the current menu, moves to the target item, and confirms the selection by releasing the mouse button. This sequence, called a task, was repeated 60 times by each subject. Each subject saw 6 sequences of 10 menus each. In each ten menu sequence, the menu type was the same, either Pie or linear, and since there are only 5 menus per task type, each menu appears twice in the sequence. The 10 menus sequences correspond to the cells in the experiment table design. Each subject performed a sequence for all 6 cells and in random order. 60 data points were collected per subject. A total of 26 subjects performed the experiment for a total of 1560 data points.
(1) The target is displayed to the user by the computer. The "x" marks the current cursor location.

(2) The user invokes the menu by pressing and holding any mouse button, in this case a pie menu, and the menu appears with the cursor location unchanged and at the center of the menu.

(3) With the mouse button still depressed, the user moves the cursor with the mouse towards the target NW which highlights to indicate selection.

(4) The user releases the mouse button to confirm the selection, the menu disappears and the process begins anew.

Figure 8: A typical task session for the Pie experiment.
For each task, the time from the first mouse button down to the correct target selection is the seek time for the item. If the user selected the wrong item, this time is included in this interval. The number of errors made as well as the sub-interval times when errors are made is recorded during the experiment by the system.

3.4. Results

Figure 9 displays the target location by item, plotted against the mean seektime. The mean seektime across target location for Pie menus is fairly constant. As expected for linear menus, the mean seek time increases proportional to the distance of the target from the initial cursor location. There was a slight learning effect for both Pies and linear menus, but Pies were consistently faster. This can be partially attributed to the constant item distance advantage

Figure 9: Target location (x) vs. seektime (y) in seconds.
of Pie menus, and partially to the small motion necessary to make a Pie selection.

Figure 10 plots the number of menus seen against the mean seek time for all menus that appeared at a particular point in the sequence of 60 menus. The region between pie and linear menu seek times seems fairly constant. It may be reasonable to assume that for this configuration of 8 items on a pie menu, linear menu performance will always be slightly slower (about 1.5 seconds). This can be attributed to the constant item distance advantage of pie menus.

Figure 11 displays the target location plotted against the number of errors. Pie and linear menus seem to suffer from a similar phenomenon—errors are made more on items half-way down the list. These are the items with the most interaction with neighboring items[Card 82].

The difference between Pies and linear menus, for both errors and times, was very significant (at better than the 0.0001 level, according to SAS ANOVA and Tukey analysis).

![Graph showing seektime (y) vs. time menu was seen by subject.](image-url)
3.5. Discussion of Experimental Results

This description of the experiment is greatly abbreviated, and so also is our discussion. See the full paper on the experiment for complete details and discussion[Callahan et al '87]. The most basic limitation of this experiment is that it only addresses menus consisting of 8 items. Secondly, it does not offer an evaluation of the basic disadvantage of Pies, increased usage of screen real estate. There were no nested menus in this experiment. And there is the question raised by the almost even division of subjects between Pie and linear menus in subjective satisfaction. Many claimed it was difficult to home in on a particular item because of the region of activation sensitivity of the Pie menu design.

One continuing issue with Pie menus is the limit on the number of items that can be placed in a circular format before the clutter or size of the menu window is impractical for
implementation of an application as a Pie menu. Perhaps, like the limiting factors active in linear menus concerning their lengths, Pie menus reach a similar breaking point beyond which other menu styles would be more useful. Additional experiments could determine where, if anywhere, this point is.

4. Applications and Implications

Pie menus are not just another way of showing a menu. They offer an additional degree of information, the radius of selection, and open the door to some rather different interaction styles. In this section we speculate on some future uses of Pie menus.

Pie menus naturally extend to selecting from hundreds of linearly items. We have actually used a Pie menu with 360 items to select a value; Figure 12 is an example of an very natural 60 item menu. This enables menus to play the role that must normally be filled by

---

Figure 12: Sixty item pie.
some completely different interaction formats, such as graphical sliders.

Pies menus raise the question of a philosophy of convenient directions. A selection's position relative to other selections is an important factor. A selections could be placed opposite of one that complements it, and 90 (or whatever) degrees away from ones that are orthogonal to it. (Figure 2 is an example.) There may be other ways to make selection directions easy to remember and use.

Nested Pie menus tend to leave a trail on the screen (see Figure 13), and an equivalent kinaesthetic 'trail' in the way the user's arm moves. This could make nested Pie menu selections must easier to remember, using 'muscle memory', than an equivalently nested linear menu. Experts could simply move ahead, while novices could wait to look at each menu. The authors' own experience is that mousing ahead to nested menus is considerably easier with Pies. In addition to mouse ahead, the good kinaesthesia of Pies means a user can be looking at something in one window while traversing menus in another.

A Pie menu can return both direction and distance information about a selection. A number of applications could take advantage of this; generally anything which can use two dimensions of input. A few examples are:

(1) Specifying direction and distance in a flight simulator.

(2) Simultaneously selecting a font style (direction) and size (distance).

(3) Simultaneously selecting a window type and size.

5. Conclusions

Pies are seem to be an exciting new development in menu design. Against their single major disadvantage of increased screen real-estate, they offer the following advantages:
(1) More information returned (item and radius).

(2) Much less mouse motion necessary.

(3) All items are nearby and equidistant.

(4) Variable (and increased) precision of selections.

(5) Selecting a Pie menu item is twice as fast as selecting a linear-menu item (experimentally measured).

(6) Selecting a Pie menu item results in one-fourth as many errors as selecting a linear-menu item (experimentally measured).

(7) Compatible with existing programming interfaces to menu systems.

(8) Relatively easy to implement.

Acknowledgements

Ben Shneiderman provided advice on experimental design. Linda Weldon helped with statistics. Glenn Pearson and John Chin were generally helpful with menus and experiments.
Perlman 84.


Schultz & Curran 86.


Shneiderman 86.

Ben Shneiderman, Designing the User Interface, Addison-Wesley Publishing Company (1986).

Sun 86.


Xerox 84.


Xerox 85.

References

Adobe 85.


Callahan et al '87.

Jack Callahan, Don Hopkins, Mark Weiser, and Ben Shneiderman, “Experiments with Pie Menus,” *in preparation*, ()

Card 82.


Dray et al 81.


Gettys 86.


Gosling 85.


McDonald et al 83.

List of Figures

1. Linear menu
2. Flight simulation compass menu
3. Nested Pies, both stack-of-cards and walking.
4. A pull-down Pie.
5. A Pie menu as it appears in the X Window system, within rectangle.
6. A Pie menu in a Sunview screen.
7. Pie menu activation region.
8. A typical task session.
9. Target location (x) vs. seektime (y) in seconds.
10. Seektime (y) vs. menu location.
11. Target location (x) vs. number of errors (y).
12. Sixty item Pie.
13. Trail of nested menus.