Enhanced Pie Menu Widget

Luis F. Cruz and Ivan Kirillov
CS6456 User Interface Software
College of Computing
Georgia Institute of Technology
801 Atlantic Drive, Atlanta, GA 30332
< luiscruz, ikirillov3 >@gatech.edu

ABSTRACT

Pie menus, unlike linear menus, are generally recognized as having good Fitts' law properties, with their menu items being placed at an equal radial distance from the center of the menu. Unfortunately, they are not widely used today, perhaps due to certain limitations, but we believe that their relative advantages can prove useful as a replacement for traditional, linear pop-up menus, particularly when augmented to be more intuitive. Accordingly, this paper discusses the development of our pie menu widget, created in Java Swing, which incorporates hierarchy, animation, and mouse-wheel based rotation, in an attempt to extend the usability of the standard pie menu paradigm.

ACM Classification: H5.2 User Interfaces: Interaction styles.

General terms: Design, Human Factors

Keywords: menus, pie menus, user interface, directional selection.

INTRODUCTION

Background – Pie Menus

Although not commonly found in real-world applications, pie menus have been shown to perform better than traditional linear menus under certain circumstances, such as when used in systems that rely heavily on non-standard input, like mobile devices [1]. The commonly accepted advantage of pie menus over linear menus can be explained best by application of Fitts' law [2], which states that the time required to move to a target area is directly proportional to the distance and size of the target.

As such, in a standard linear menu, the items are all the same size and laid out in a single direction, which means that the distance traveled is directly related to the height of each menu item. However, in a pie menu, all items are radially equidistant from the central point of the menu, but instead lie in separate directions. Thus, whereas a linear menu can have items that are considerably distant from the cursor, which makes navigating to them take considerably longer than others, the time required to move the cursor to a pie menu item is identical for each item, making item selection faster than in a linear menu.

Developmental Evolution

Hierarchical Menus. This key property of the pie menu, along with the generalized notion that physical direction is easier for a user to remember than list order, are the main reasons why we chose to develop a pie menu to serve as a possible replacement for the traditional linear pop-up menu. However, one notable problem we initially discerned with this approach is the fact that a traditional pie menu can only display a limited number of items (with 8 being commonly accepted as the maximum [3]) without adversely affecting its ease of use and overall utility. Since a linear menu can easily hold more than 8 items, we needed a structure that would allow our pie menu to hold more information while still retaining the benefits of the pie menu paradigm itself.

After reviewing related work, particularly a paper done by P.J. Lyons and M.Pitchforth on custom oval widgets [4], it became clear that perhaps the best approach to solve this limited display problem would be to incorporate tree-based levels of hierarchy into each main pie menu button (with this main button serving as the root of its own tree), thus allowing the menu to display far more than 8 items. Hierarchy is also beneficial because it allows us to hide the secondary menu until requested by the user, which decreases the overall size of the menu structure itself. In order to increase development simplicity, we decided that our pie menu would have 8 initial pie buttons, and with each such button having one level of hierarchy. In terms of hierarchical positioning, it was decided that a concentric-based layout would be appropriate, because it essentially offers the same affordance as the initial pie menu, while making hierarchical item placement relatively straightforward.

Of course, having these hierarchical menus brought up another problem, that of information overload. Any benefit to having such a pie menu would be negated if the user has to spend a significant amount of time sorting through the hierarchical menu to find what they are looking for. As such, we decided that in order to make the hierarchical menus highly functional, we would have to limit both the number of total hierarchical items (i.e. the leaves) as well as the number of items visible at any one time. Therefore, we chose to have no more than 12 total hierarchical menu items for any pie menu button, and with no more than 7 being visible at any one time for quick target acquisition by the user.

Hierarchical Scrolling. Making a limited number of hierarchical menu items visible at any one time introduced the issue of allowing the user to easily navigate throughout the entire hierarchical menu. Initially, we determined that the best course of action to take in this regard would be to have unique arrow buttons situated on both sides of the concentric hierarchical menu, where the act of clicking on an arrow would rotate the hierarchical menu one item in the direction of the arrow. However, we quickly realized that such a solution, while offering a relative amount of affordance, would make going through the entire hierarchical menu a cumbersome experience, since the user would have to click 11 times to cycle through the entire menu. Also, the positioning and rotation of each arrow would have to be determined by the angular positioning of the each main pie menu button, adding an extra layer of complexity to our design.

At this same juncture, we determined that a better, more elegant solution to this cycling problem would be to utilize the now ubiquitous mouse scroll wheel. We figured that the use of the scroll wheel would be much more natural to the user than the clickable arrows if implemented properly, so it was decided to implement mouse wheel scrolling of the hierarchical menus in both directions, with an upwards scroll cycling the menu counter-clockwise and a downwards scroll cycling it clockwise. Likewise, since it is easy for a user to vary their relative mouse scrolling speed, we decided to set the hierarchical scroll increment to be less than a single hierarchical menu item, which means that it takes more than one scroll movement to rotate the menu by a single menu item. However, this also provides a sort of buffer so that the user cannot easily scroll too quickly through the hierarchical menu and miss the menu item they are looking for.

Animation. Beyond window sizing and re-sizing, animation is not commonly found in today's user interfaces. However, it has been shown to have certain positive effects when wisely utilized, including being able to provide an effective transition between visual states by removing large and unexpected changes, as discussed by Chang and Ungar [5]. Thus, since we determined that the user would not typically be aware of the hierarchical menus that reside beyond the initial pie menu buttons without prior use, we decided that it could be beneficial to include some sort of animation for the expansion and collapse of these hierarchical menus. In theory, such animation could provide better feedback to the user by showing the expansion and collapse of the hierarchical menus as a quick progression, rather than an instantaneous event. Of course, such animations would also have to be quick and seamless enough so that they do not hinder or slow down the performance of a more experienced user of this pie menu.

Menu Item Representation – Text vs. Icons. The final design decision we had to make was whether to use traditional text labels or simple icons to represent each menu item, regular and hierarchical, in our pie menu. While it was obvious to us that using text labels would make the pie menu more akin to a linear pop-up menu, we realized that this choice would bring a number of additional problems, the first being the problem of orientation based on text sizing. Text-sized based orientation is never a problem in linear menus, since all menu items are horizontally aligned and sized so that there is enough room to fit the corresponding text label.

In our pie menu, at least in the initial, non-hierarchical menu, text-sizing based orientation is also not a significant problem, as we could determine the minimum radius needed to fit the largest text label and draw the pie menu accordingly. Of course, since this increases the size of each menu item area in order to fit the largest text object, it could potentially create a very large pie menu structure that would take up far too much screen space to be considered useful.

Accordingly, in the hierarchical menu, this could also prove problematic, as the distance of the hierarchical menu from the main pie menu, as well as the size of each hierarchical menu item, would be dependent on the length of text in each menu item. As such, each individual hierarchical menu would vary in size and appearance, thus creating a non-uniform look and feel to the hierarchical menus that would hinder the affordance offered by having uniformity throughout the entire menu structure.

Thus, based on this reasoning, it was decided to utilize icons for both initial pie menu items as well as the hierarchical menu items. While this approach may not have been completely in-line with our ultimate goal of developing a pie menu replacement for the linear pop-up menu, it still allowed us to develop our test pie menu widget while focusing on the other, more important mechanics such as hierarchy and mouse-wheel scrolling.

ARCHITECTURE OVERVIEW

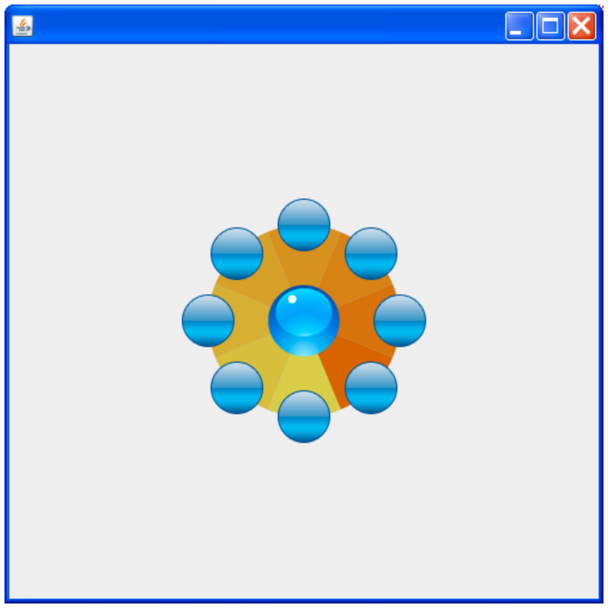

Figure 1: A screenshot of the initial state of the PieMenu Widget.

This Pie Menu Widget was designed by creating two main MVC components: the PieMenu and PieButton Java classes. These two classes are designed to be composed of 3 individual components, which are the Control component, the UI (View) component and the Model component, thus giving a total of six Java classes.

The PieButton component is in charge of the root button and its hierarchy buttons, building all of them from the JButton Java Swing library. Using a JButton was beneficial in that it allowed us to inherit all of the capabilities of a real button and also allowed us to apply a custom skin. The PieButton component has the following responsibilities:

- Build itself as a root button and build its hierarchy buttons.
- Set the correct visibility of its hierarchy buttons.
- Set and update the correct position of the buttons.
- Update the angle of each of its hierarchy buttons.
- Control the animation of its hierarchy buttons.
- Keep track of the current states (is it is expanded, contracted, if is in a transition).

The PieMenu component is the main component and the one that has to be created by the application developer if the PieMenu Widget is to be used. As such, the PieMenu has these responsibilities:

- Instantiate the root menu buttons (PieButton instances).
- Position the root buttons.
- Update the angle for the root buttons.
- Build the arcs (Java Arc2D) in the inner circle.
- Paint the inner circle (Figure 1- the colorful circle with the blue circle at the center).
- Paint the arcs (Figure 1- Each color segment in the inner circle is an arc).
- Paint the “tree” lines that join the root button with its hierarchy buttons (Figure 3).
- Handle the mouse pressed listener to trigger the root buttons (Figure 2).
- Handle mouse moved listener to activate the root buttons (Figure 2).
- Handle the mouse-wheel moved listener, and pass the responsibility to update the buttons to itself (in case of the root button’s rotation) or to the PieButton objects (in case of the hierarchy button’s rotation). (Figure 5).

The reason we decided to handle the painting and listeners inside of the PieMenu component was due to the complexity of having each root menu (PieButton) in possession of its own rendering area. We initially designed it in this manner, but we started to experience problems with Listener dispatching and drawing due to the proximity of the buttons, which we later determined were caused by the overlap of these components.

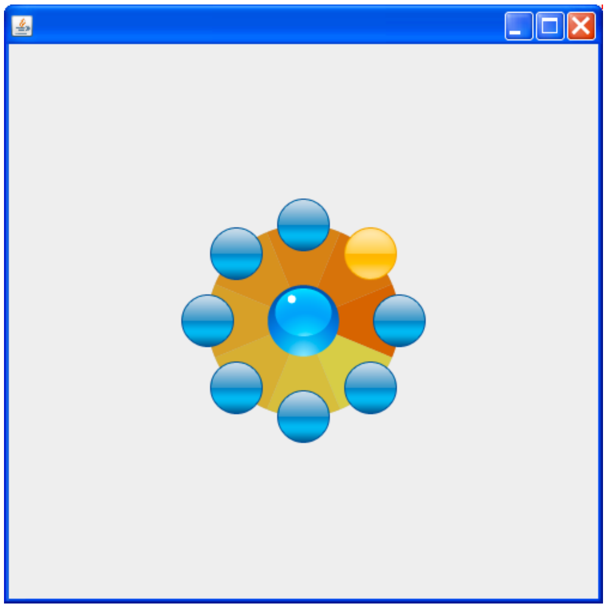

Figure 2: A screenshot of the root highlight button state.

Hierarchy

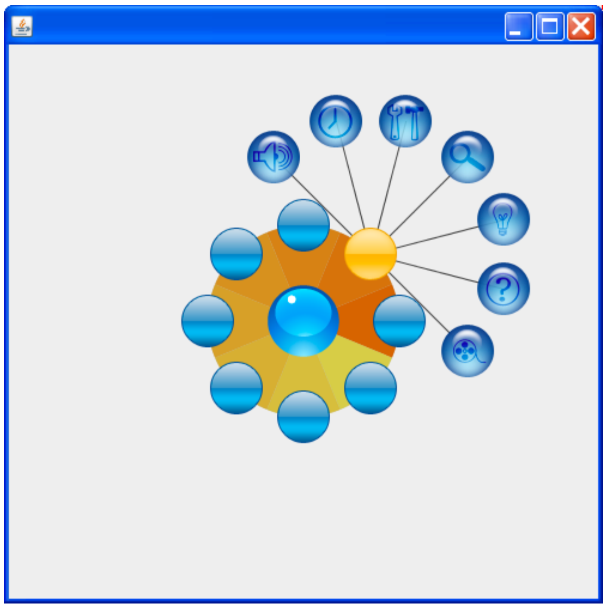

Figure 3: A screenshot with the root button triggered with its hierarchy buttons expanded.

Our hierarchy implementation was essentially created by modifying the main pie button class, PieButton. Inside the class, we have a simple array of type JButton, and a method which populates this array with the corresponding JButtons and their icons. Although this method supports any integer number of buttons as its input parameter, it is currently hard-coded to 12 inside the main PieMenu class when called, due to the fact that we only created 12 distinct icons to populate the hierarchical menu items with.

In an almost identical manner to the PieButtons themselves, the positioning of the individual hierarchical buttons contained in each PieButton is accomplished through the use of parametric equations. Our updateHierarchy() function determines the angular spacing by simply dividing 360 degrees by the number of buttons, and then loops through the JButton array in order to determine the correct X,Y coordinates for each button, using sin and cosine. While still in the loop, we set the location of each hierarchical button based on these X,Y coordinates and the offset of the parent PieButton, using setBounds(). Since setBounds() only takes integer values for its positioning parameters, while sin and cosine often do not give us integer results, we must first cast our double value coordinates as integers, losing some precision in the process, but not enough to give the impression that our buttons are imperfectly spaced in a concentric fashion.

Perhaps the biggest challenge we encountered when creating these hierarchical buttons was how to optimally hide and display only the buttons that we wanted. However, since the buttons themselves are laid out concentrically, we determined that this could also be accomplished using a fair bit of trigonometry. In this case, we have a method called setHierarchyVisible(), which is also responsible for setting some of the animation timer variables based on the current state of the hierarchical menu.

Inside the method, we utilize a loop to go through the JButton array and first determine in which of the four 90 degree quadrants the hierarchical button's parent PieButton resides. Once this is determined, we calculate the angle between the parent PieButton and the hierarchical button using atan2() in order to determine if the button lies beyond the 180 degree arc that extends outward from the PieButton, with separate logic for each quadrant. If the hierarchical button is inside the arc, then we simply set it visible, but otherwise we do not, all of which is accomplished by calling the setVisible() method of each hierarchical button. Initially, we had a problem where buttons near the edge of the 180 degree arc would be incorrectly set invisible, but we determined that this was a result of the our previous X,Y coordinate truncation in the positioning logic as a result of the double to integer conversion. To fix this, we simply padded the angular logic for each quadrant by a single degree in each direction.

Animation

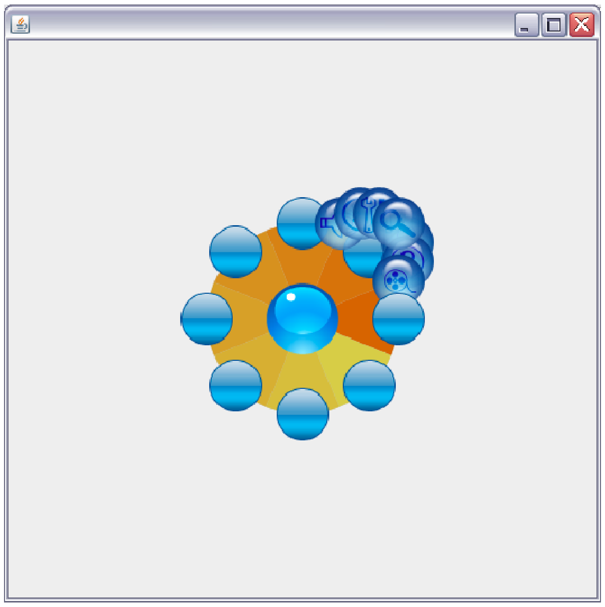

Figure 4: A screenshot of one of the initial frames of the expansion animation.

We have implemented linear animation; that is, we increase or decrease our animation parameters by a constant value. A more realistic animation would have been the implementation of an "easy-in" and "easy-out" animation such as the techniques described by Thomas and Calder's paper [6].

Our Pie Menu Widget has two animations. On both animations we are using the *javax.swing.Timer* Class because of the use of swing classes; therefore, we must do it on the event dispatch thread. The first animation is when the Pie Menu Widget appears, and for this animation we setup the Timer object so that it iterates throughout each call of setVisible() set to true for each of the root buttons. The second animation is the one for the expansion and contraction of the hierarchical buttons; this animation was achieved by creating a Timer object that depending on the actual state of the hierarchy (controlled by booleans) it decreases or increases the radius being used in the formula for the positioning of each respective button. A call to the method that updates this location is triggered by a *mousePressed* Listener.

Mouse-wheel Rotation

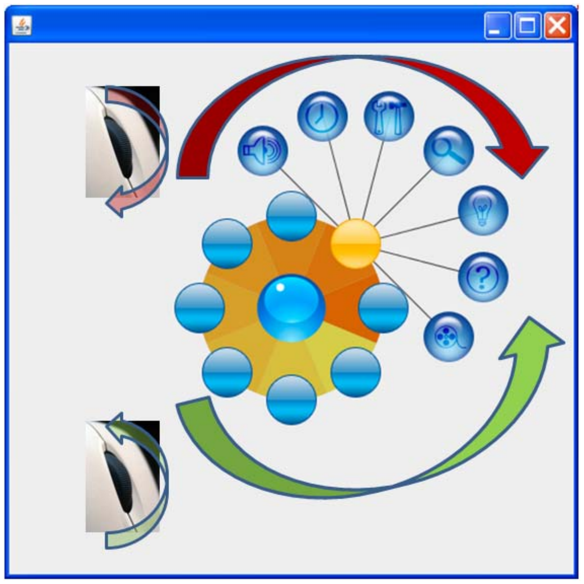

Figure 5: Representation of the hierarchy buttons rotation triggered by the mouse wheel.

There are two rotation events in our Pie Menu Widget; one is at the root component which rotates the whole widget as a unit, and the other one rotates the hierarchy buttons. Both of these rotations are handled by the same *MouseWheelListener* Java event. We split both rotations by adding a Rectangle Java object as a bounding box for the inner part of the Widget (the area with the multi-colored arcs), then we check if the mouse coordinate is contained inside this bounding box, and if it is we then increase or decrease the angle for the root buttons; otherwise, we check every root menu if it is expanded, and if it is we then increase or decrease the angle for the hierarchical buttons of the expanded root button. Both cases end up in a change of a parameter (an angle) and the redrawing of the affected buttons.

FUTURE STEPS

The most important feature for future development is extending the implementation of hierarchical pie menu so that it can expand to multiple levels. Currently, our pie menu widget can expand only up to two levels: 8 buttons for the root level and then 12 buttons for the leaves, giving a total of 96 menu options. Ideally, we will need to refactor our widget architecture so that it can efficiently handle a real tree structure and thus display multi-level hierarchy menu, that is 8 x 12 x 12 x ... x n.

There is still more room to improve Fitt's Law; although we implemented an "interaction" area within the circle for the root level so that the user does not have to move the cursor to the target area, we did not implement the same feature for further levels - in this case, the user has to move the cursor to the button. Going beyond adding this to the other levels, will include the implementation of the concept of Marking Menus [7], which will greatly improve the user experience and performance by integrating the mouse movement to the menu expansion and rotation.

The architecture will have to be improved and extended in order to provide a more customizable Java component with an API that allows any application developer to have more control over this widget without modifying its core architecture such as deciding the number of levels, labels, and being able to accept a Java treeModel; moreover, it would be very helpful if the user can also control the size of the widget components, and for this our widget will have to work with SVG images so that they can be resizable while maintaining a good image quality.

In terms of having a pie menu analogous to a linear pop-up menu, we would ultimately need to incorporate some form of text menu items. While this would not require much modification for the main pie menu buttons, we would need to create a new class for the hierarchical buttons in order to ensure correct orientation and positioning of their text labels. Accordingly, we would need to modify the hierarchical button display logic so that there would be proper spacing and no possible interference between the main pie menu and the hierarchical buttons.

We think that the animation we implemented in this phase suffices the requirements for a pie menu, and further animations can actually clutter the widget and even adversely affect performance.

CONCLUSIONS

After going through this development process, it is hard for us to justify the replacement of traditional, linear pop-up menus with pie menus, augmented or not. Even discounting the lack of available pie menu libraries in most user interface development kits, it is clear that pie menus have too many quirks of their own for a developer to consider seriously such a proposal. Most notably, pie menus still take up a substantial amount of screen real-estate, even when implemented entirely with icons such as in our case. Likewise, implementing any form of text menu items in a pie menu can be a troublesome process, and one that is unlikely to be worth the relative navigational advantage that pie menus can provide.

On the other hand, pie menu widgets appear to be one of the best tools that can easily combine the properties of contextual menus and floating palettes. Our pie menu widget would be an excellent option for a tools palette given the ease with which one can skin the buttons and implement interactions such as scrolling; moreover, we found that our Pie Menu Widget works better in a menu structure that can be represented with icons instead of text due to the aforementioned size constraints; therefore, applications for graphics and multimedia are those that can best take advantage of our widget.

ACKNOWLEDGEMENTS

We wish to thank our professor Keith Edwards and assistant Ben Davison at Georgia Tech for all the help, recommendations and advising provided to us during the development of this project.

REFERENCES

1. Vaittinen, T., Viljamaa, T., and Piippo, P. 2007. Design issues related to pie menus for 5-way joysticks. In *Proceedings of the 4th international Conference on Mobile Technology, Applications, and Systems and the 1st international Symposium on Computer Human interaction in Mobile Technology* (Singapore, September 10 - 12, 2007). Mobility '07. ACM, New York, NY, 564-571.

2. Callahan, J., Hopkins, D., Weiser, M., and Shneiderman, B. 1988. An empirical comparison of pie vs. linear menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Washington, D.C., United States, May 15 - 19, 1988). J. J. O'Hare, Ed. CHI '88. ACM, New York, NY, 95-100.

3. Rick Komerska, Colin Ware. A Study of Haptic Linear and Pie Menus in a 3D Fish Tank VR Environment. In *Proceedings of IEEE HAPTICS 2004.*

4. Lyons, P.J.; Pitchforth, M.; Page, D.; Given, T.; Apperley, M.D. The Oval menu - Evolution and Evaluation of a Widget. In Computer-Human Interaction, 1996. Proceedings., Sixth Australian Conference on 24-27 Nov. 1996 Page(s):252 - 259 Digital Object Identifier 10.1109/OZCHI.1996.560018.

5. Chang, B. and Ungar, D. 1993. Animation: from cartoons to the user interface. In *Proceedings of the 6th Annual ACM Symposium on User interface Software and Technology* (Atlanta, Georgia, United States). UIST '93. ACM, New York, NY, 45-55.

6. Zhao, S. and Balakrishnan, R. 2004. Simple vs. compound mark hierarchical marking menus. In *Proceedings of the 17th Annual ACM Symposium on User interface Software and Technology* (Santa Fe, NM, USA, October 24 - 27, 2004). UIST '04. ACM, New York, NY, 33-42.

7. Thomas, B. H. and Calder, P. 2001. Applying cartoon animation techniques to graphical user interfaces. *ACM Trans. Comput.-Hum. Interact.* 8, 3 (Sep. 2001), 198-222.