Cocoa Pie Menu Implementation

Julian Missig
Carnegie Mellon University
jmissig@cmu.edu

ABSTRACT
This paper describes an implementation of a pie menu [3] for Mac OS X using Cocoa [4]. It discusses the issues involved in its creation, future steps for the library before releasing it to the public at large, and the author's musings on the real reasons Pie Menus are not popular.

Author Keywords
Pie menus, marking menus, Mac OS X, Cocoa implementation

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

INTRODUCTION
Fitt's Law tells us that the time to target an object on a screen using a mouse or similar input device is a function of both the size of the object and the distance to do that object. The Macintosh operating system as well as its predecessors have exploited this: The main menu was placed such that the items could be selected at the absolute top edge of the screen. This gives the menu items an essentially infinite size in one direction, allowing for easy selection by a flick of the mouse in the general direction of the item. Thus, the menu item is easier to select and can be selected much faster.

Linear menu items are almost the antithesis of an exploitation of Fitt's Law. All of the items are the same size, and they are all laid out in the same direction. The only difference selection-wise between one item and another is the exact amount of distance traveled, which comes down to item height (the smallest of the two dimensions of a linear menu item).

In a pie menu, each menu item is a slice of a circle. When the menu is brought up the cursor is at the center of the circle. Clicking in the center dismisses the menu. With this design, each menu item differs from the other by actual direction. Distance is the same to each menu, but since each pie slice is in a different direction, the user only has to move the mouse in the direction of the item. This sounds good in theory and it bears out in practice: Users can select items from pie menus faster than they can select items from linear menus [3].

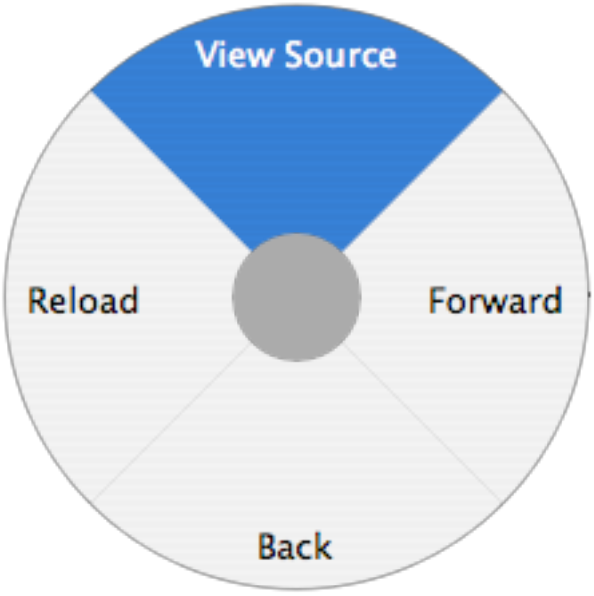

Figure 1. A four item pie menu

In theory users can also learn pie menus better and faster than linear menus. It should be easier for users to proceduralize and remember a simple physical direction than an order in a uniform list.

While pie menus may have many advantages, they are not particularly common in real world applications. A lack of pie menu libraries in the most common user interface toolkits does not help. I therefore decided to implement a pie menu in Cocoa, Apple's toolkit for Mac OS X, which I will release as a Framework (a relocatable dynamic library for Mac OS X). The goal is to eventually be able to replace the contextual menus in mainstream applications such as Safari or Mail. Third parties could also make use of the Framework in their own applications.

Throughout this paper I will use the terms "user" and "developer." The user is the end user who actually uses the pie menu in an application with their mouse. The developer is the third party programmer who makes use of my library.

ARCHITECTURE OVERVIEW

The pie menu classes I created were built to reflect the Cocoa menu API, `NSMenu`. For reasons discussed later, I was not able to subclass `NSMenuView` and simply replace that.

The architecture is broken up into two classes which are intended for the developer, `JMPieMenu` and `JMPieMenuItem`. The other two classes, `JMPieMenuView` and `JMPieMenuWindow`, are used by `JMPieMenu` to do the actual work of displaying the menu.

`JMPieMenuItem` keeps a lot of information about itself. It knows its rotation from zero degrees, the slice angle (angle of the circle it takes up), its title, and its drawing path. The view asks each item for this information as it draws.

WHAT WAS EASY

There were a couple of things which were easier than I anticipated. I did not know Cocoa (or even Objective-C) before I started, and once I wrapped my head around some of its idiosyncrasies, it was rather easy to rapidly put together code. It is impressive how little code is required to do a lot of things.

Along those lines, the actual act of drawing a pie with slices, even when those slices are each actually separate drawing paths, turned out to be much easier than I expected. I found out about Cocoa's `NSBezierPath`, spent a little bit of time learning how it works, and then it was not very long before I was up and running with a basic circle. I spent more time dividing the drawing up into several functions and figuring out which functionality and information should be kept in `JMPieMenuItem` than I did writing the initial code which generated the proper paths.

One important aspect of the project from an integration perspective was making sure that the menu looked and felt like a native Mac OS X menu. This too turned out to be easy. There are `NSColor` functions available which simply supply the background color for controls and the highlight color for menu items. It even provides the proper colors when using Graphite instead of Aqua.

Once I had the general class architecture set up, it became very easy for me to just work on the harder problems without worrying about the interactions between classes. There was a long stretch of time where I was only working on one function in `JMPieMenuItem`, then another stretch where I was just focused on a few functions in `JMPieMenuView`. This architecture seemed to work extremely well for the problems I encounted.

IMPLEMENTATION ISSUES

A large majority of my time on the initial implementation of this project was spent on a small number of issues. There were of course a lot of smaller issues, but they were and are more easily solved, and fortunately even in sum they did not seem to add up to very much.

Sizing the Pie

Because one of the major goals was the replacement of contextual menus in Mac OS X, the pie menu had to be text-based rather than icon-based. An icon-based layout would have been simpler because all of the icons could be the same size, allowing for consistent pie menu sizing and easy layout. The RadialContext pie menu plug-in for Firefox [8], for example, is icon-based. The text labels can be outside the pie while the user learns the icons.

Speed of reading is reduced when text is rotated, drawn on a circle, or skewed, so I decided to keep my text horizontal. This too added to my problems. I wanted all of the item titles to fit within their respective pie slices.

In order to figure out the size of the overall pie, `JMPieMenuView` iterates through each `JMPieMenuItem` and sends a message asking for the minimum desired radius. The `JMPieMenuItem` calculates this value by making use of similar triangles. A small triangle is generated with a side of the length of the inner circle of the pie and three angles based on the angle of the slice and its rotation from zero degrees. The other angles of this triangle are known since we know the rotation, the fact that the text is horizontal, and the slice angle. From this small triangle we use a ratio of sides to get the ideal size of a larger triangle which includes the full width of the item title. It took me a while to come up with this solution, and even longer to get the math right.

Once I finally figured out the sizing, I realized I had to figure out the position of the text in the actual pie menu. The difference is that the actual pie menu probably has a radius larger than the minimum radius the pie menu item requires (because the view went through and picked the largest radius of all items). Before thinking about it too hard I realized I could reuse the similar triangle code to determine position as well.

NSMenuView's Disappearance

In an ideal world my code would simply be a drop-in replacement for `NSMenu`. Apple used to make use of an `NSMenuView` for displaying `NSMenu`. I could have simply subclasses `NSMenuView` and `NSMenuItem` and had easy code to replace a context menu wherever a developer pleased.

Unfortunately Apple no longer uses `NSMenuView` in Cocoa [6]. It seems Apple optimized the `NSMenu` display for its rectangular items. This meant that I had to make my own classes with very similar protocols to `NSMenu`'s.

Optimizing Drawing

Currently I am using Cocoa's `NSBezierPath` to draw the pie menu and its slices. This has the advantage of being

clear, easy, and generally the right way to go about this kind of simple drawing.

Unfortunately, it is nowhere near as efficient as is needed for really fast-displaying menus. Users have already reported that sometimes if they move their mouse quickly the drawing lags behind the cursor movement. I spent some time looking through my code with a performance tool, and more than 80% of the time drawing is in a single call to tell Cocoa to draw the `NSBezierPath`. This is not Cocoa's fault; the paths are a complicated series of points which are rotated and drawn many times.

I optimized my loops everywhere that I saw I could, and even cache all of the paths and transforms, but for optimal performance some other kind of drawing will need to be done. Even Apple did not seem happy with standard high-level drawing for its `NSMenu` [6]. Optimizing drawing is made more difficult with the fact that the pie menu slices simply are not rectangles, so images of them cannot be easily cached unless they contain alpha channels.

FUTURE STEPS

There are still many things my pie menu implementation needs before it is a good, solid, general-use pie menu Framework for Mac OS X.

There are a series of quick little things that need to be done before a public release. `JMPieMenuView` currently does not draw arrows for submenus. This should be relatively easy to fix. The APIs for `JMPieMenu` and `JMPieMenuItem` should be cleaned up and made to match their `NSMenu` counterparts more closely. There are a lot of short convenience messages which need to be implemented to match `NSMenu`'s API.

Currently my pie menu does not support click-and-drag selection like a regular Mac OS X menu. This stems from the fact that `JMPieMenu` creates a window on `mouseDown`. All of the `mouseDrag` events between a `mouseDown` and a `mouseUp` are sent to the originating window; which happens to be the developer's window rather than my generated menu. I will need to figure out how to get the `mouseDrag` events to my window so I can make use of them.

I did not yet get to test replacing Safari or Mail's contextual menus with JMPieMenu. It seems that this should be possible by creating a Cocoa Bundle which hijacks the messages they pass and generates a JMPieMenu instead of the NSMenu it is expecting. I do not expect this to be *too* difficult, but it will take a little bit of work.

`JMPieMenuWindow` is a fullscreen window. This makes it easier for me to grab the `mouseMoved` events outside the window to track the cursor, and it makes it easier to simply place the pie at a location in the view which matches

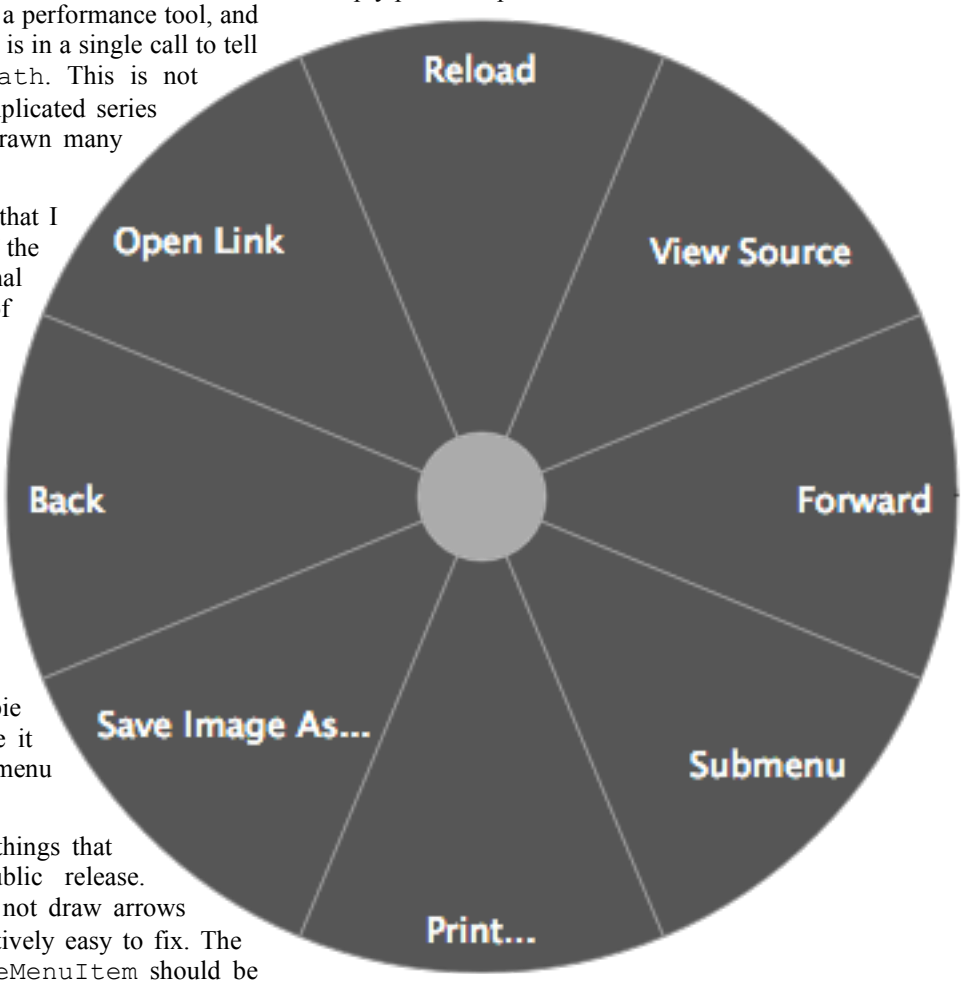

Figure 2. The "advanced" look

a location on the screen. The slowdown due to this fullscreen window is not as much as expected. Still, ideally this window would fit the size of the pie (which means the window will have to be sized to the calculated size of the pie).

Speaking of grabbing `mouseMoved` events outside the window, since my pie menu does not support click-and-drag selection I implemented a system in which the user can simply move their cursor outside the pie to select an item; no clicking is necessary. After a 200 millisecond delay, the item the cursor has radially selected is automatically activated. With click-and-drag selection, this design choice should be reconsidered. It is nice for trackpads, where it is difficult to click-and-drag select, but user error may be too large of a problem. My hope is that the large pie area around the cursor is enough of a "buffer" that automatic

selection outside of the pie is a feature advanced users can make use of, rather than a problem that novice users have difficulty with.

This pie menu draws an idealized marking line which displays the path the user took to select submenus and submenu items [7]. This marking line can appear somewhat confusing and should probably be animated, rather than displayed all at once.

There are also a few aesthetic issues which I believe should be fixed in future versions of `JMPieMenuView`. The largest is still text layout. Currently text layout is done in a strictly bottom-up fashion; each `JMPieMenuItem` reports what it thinks is best, and that is what is drawn. I think that some top-down information could help lay out the text in a nicer fashion, but figuring out the mechanics will be difficult. `JMPieMenuView` would have to figure out the locations that each item wants for text, the locations that are possible, and select a subset of those locations as ideal, then draw those.

The Mac OS X system look of the pie menu also may not be what modern Mac OS X developers want. In fact, it is more likely that developers of applications who want to use my pie menu will want it to have an "advanced" or "Pro" look (see Figure 2). I will offer this look as an optional setting on `JMPieMenuView`.

Finally, `JMPieMenu` should be wrapped up as a dynamically loaded Framework which can simply be linked by the application which wishes to use it.

WHY PIE MENUS ARE NOT POPULAR

Don Hopkins, one of the early advocates of the pie menu [3], explained in an interview with an online magazine why he believes that pie menus are not more common [9].

In summary he believes that it is simply too difficult to extend the existing user interface toolkits. He believes that application developers do not develop new controls and thus would never think to develop a pie menu. He also argues that Apple and Microsoft are simply lazy and do not like to implement things they did not invent [9].

In short I do not believe his arguments. It was not exceedingly difficult for me to extend Cocoa, and I did not know Cocoa before I began. In my experience application developers write new controls all the time; successful applications are based around a set of unique controls. I cannot speak for how lazy Apple and Microsoft may be and how much they suffer from "Not Invented Here Syndrome," but in Apple's recent history they have created things like Exposé and Dashboard [2, 1] and Microsoft has created Command Tabs in Office 12 [5]. Both have been quite willing to copy user interface innovations from one another.

While I agree with Hopkins' conclusion that it is easiest to expose users to new user interface ideas through games, it seems to me that Hopkins' arguments do not get at the key issues.

It is really hard to make pie menus look aesthetically pleasing and maintain the usability and readability gains that they were designed to bring. It is quite easy to simply rotate text or bend text and have a pretty pie menu, but the resulting menu is not as easy to read and may not be as fast as a similar menu with horizontal text. It is also easy to do away with the pie altogether, as has been suggested for advanced marking menus [7], but I do not believe these menus are anywhere near as immediately clear and obvious to new users as pie menus are. One of the great features of pie menus is that new users and advanced users alike can appreciate advantages from their use.

Circular math is not easily translated into the rectangular drawing and layout systems used by toolkits like Cocoa. It is not that trigonometry in and of itself is difficult, but it is quite difficult to optimize the layout and display of text when it is necessary to keep a large number of `floats` around to do the proper math. There are many bounding issues to worry about in rectangular space, and a pie menu implementation requires that you also worry about things in circular space.

Pie menus are also limited in number of items. There are a lot of extremely long menus in Mac OS X which cannot be easily translated into small pie menus. Ideally a pie menu should only have four or eight items. Six works somewhat well. This is a major limitation for a developer hoping to use them; the developer has to break down his menu structures into submenus which work with those numbers.

Even with this limitation on number of items, any text-based pie menu will become quite large and take up quite a bit of screen space. I imagine it is difficult to convince a developer that the screen space trade-off is worth the advantages a pie menu offers.

For these reasons, the easy places to begin implementing pie menus would not be the place I picked. Ideally a developer could have an icon-based pie menu, like RadialContext [8]. This makes the item titles much less important and allows for focus on other aspects of the implementation. Even sizing becomes easy.

CONCLUSION

While there are reasons to believe that my approach of simply replacing contextual menus in existing applications is not the right direction for a pie menu implementation, I still feel this implementation is worthwhile for third party developers. The work that needs to be done to finish up this implementation is not a lot compared with the work done to start it. This is thanks to a well-separated architecture which allowed me to get through the harder problems more quickly.

Most of the developer issues with pie menus are in the implementation, not the use of a Framework. Developers

will have to make smart decisions about which items appear, the length of the item titles, and their positions, but I have done all the trigonometry for them. Pie menus are a lot of fun and I hope to find this implementation used in third party Mac OS X applications in the future.

ACKNOWLEDGMENTS

I thank my friends (they know who they are) who tested and commented on my various pie menu tests, even though the initial versions made their laptops hot or their PowerMac fans spin up.

REFERENCES

1. Apple – Mac OS X – Dashboard. http://www.apple.com/macosx/features/dashboard/.
2. Apple – Mac OS X – Exposé. http://www.apple.com/macosx/features/expose/.
3. Callahan, J., Hopkins, D., Weiser, M., and Shneiderman, B. An empirical comparison of pie vs. linear menus. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems* (1988). J. J. O'Hare, Ed. CHI '88. ACM Press, New York, NY, 95-100.
4. Cocoa. http://developer.apple.com/cocoa/.
5. The New Microsoft Office User Interface Overview. http://www.microsoft.com/office/preview/uioverview.mspx.
6. NSMenuView (Objective-C). http://developer.apple.com/documentation/Cocoa/Reference/ApplicationKit/ObjC_classic/Classes/NSMenuView_index.html.
7. Tapia, M. A. and Kurtenbach, G. Some design refinements and principles on the appearance and behavior of marking menus. In *Proc. of the 8th Annual ACM Symposium on User interface and Software Technology*. UIST '95. ACM Press, New York, NY, 189-195
8. RadialContext. http://www.radialthinking.de/radialcontext/.
9. Why pie menus aren't ubiquitous? http://www.infovis.net/printMag.php?num=125&lang=2.