The Hotbox: Efficient Access to a Large Number of Menu-items

Gordon Kurtenbach, George W. Fitzmaurice, Russell N. Owen and Thomas Baudel

Alias | Wavefront
210 King Street East
Toronto, ON, Canada, M5A 1J7
<gordo, gf, rowen, tbaudel>@aw.sgi.com
1+ 416 362-9181

ABSTRACT

The proliferation of multiple toolbars and UI widgets around the perimeter of application windows is an indication that the traditional GUI design of a single menubar is not sufficient to support large scale applications with numerous functions. In this paper we describe a new widget which is an enhancement of the traditional menubar which dramatically increases menu-item capacity. This widget, called the "Hotbox" combines several GUI techniques which are generally used independently: accelerator keys, modal dialogs, pop-up/pull down menus, radial menus, marking menus and menubars. These techniques are fitted together to create a single, easy to learn yet fast to operate GUI widget which can handle significantly more menu-items than the traditional GUI menubar. We describe the design rationale of the Hotbox and its effectiveness in a large scale commercial application. While the Hotbox was developed for a particular application domain, the widget itself and the design rationale are potentially useful in other domains.

KEYWORDS: menus access, menubars, two-handed input, transparency, marking menus

INTRODUCTION

In this paper we describe the design of a menu access widget in Alias |Wavefront's professional 3D computer animation and design application, Maya [14]. Because Maya is a professional's tool it presents many challenging user interface requirements. First and foremost, Maya allows complex and sophisticated controls over 3D data and the behavior of 3D data over time. For example, Maya is used for computer graphics special effects in blockbuster Hollywood movies like "Jurassic Park" and "Toy Story". This sophisticated functionality results in an application with hundreds of commands. Professional users also require efficient access to commands since they may spend a huge number of hours operating the application under strict deadlines. Therefore even small performance improvements (like menu selection speed) can dramatically affect user efficiency and their perceived efficiency of the application. Another major design requirement for this class of application is to reduce the complexity of the user interface whenever possible. The nature of data and the operations on the data is, by itself, complex. Adding in a complicated user interface would only further increase complexity of the application.

These challenges produce three basic problems for the traditional GUI design:

Quantity: Maya has approximately 1200 commands which would be typically found in the menubar. This number of commands will increase with subsequent versions. Roughly speaking, at the very most 20 menus can be placed in a menubar that span the entire length of a high resolution screen (1280 pixels across). With 1200 commands this results in menus that on average have 60 items in them. In practice this results in information overload.

Speed: Users want fast access to frequently used commands. In traditional GUIs, hotkeys (also called "menu accelerators"), are used for the frequently used functions. In Maya, a default set of hotkeys are used to access frequently used functions, however, this allows access to only a small fraction of the 1200 commands. Increasing the number of hotkeys creates problems. First, as the number of hotkeys assignments increases the hotkey mappings become hard to remember ("why is ctrl-d mapped to "Create IK Joint?"). Second, the keystrokes themselves become slow to articulate (for example, ctrl-alt-P). We also support user-definable hotkeys, however, this still has the same limitations.

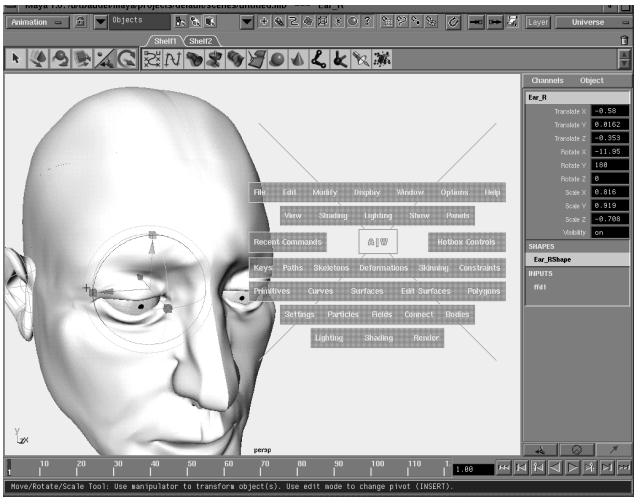

Figure 1: Maya application with Hotbox widget in the center.

Submitted to CHI'99.
copyright Alias|Wavefront 1999 withheld.
Please do not distribute without prior authorization.

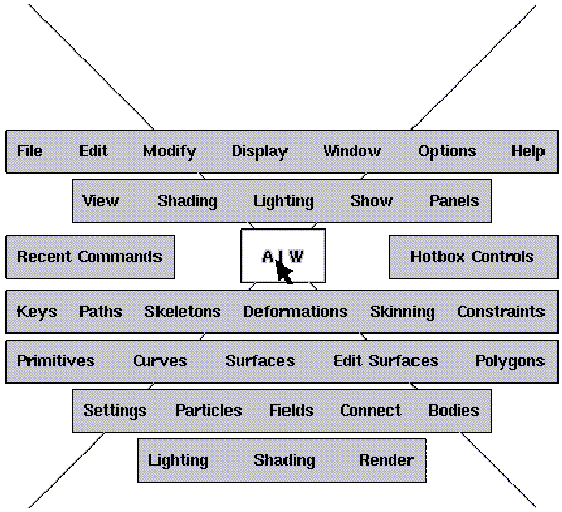

Figure 2: Hotbox widget in “Show All” rows mode.

Unification: In a traditional GUI, commands may get distributed between toolbars and the menubar. Typically GUIs place functional modes into toolbars and one-shot actions into the menus. However, from a functional point of view, some particular modes and actions are closely related and therefore should be found close together in the interface. We believe this aids the user in learning the application and in finding commands in the interface especially when there are numerous commands. Thus we developed the requirement of “unification”: modes and actions should be grouped together in the interface according to function.

We also wanted to create a menu access technique that would unify novice and expert behaviors. In a traditional GUI, novice and expert operation of the interface can be dramatically different. For example, a novice user may exclusively use only the menubar while an expert may almost exclusively use hotkeys. The radical difference between these two behaviors makes graduating from novice to expert behavior an explicit (and extra) effort. We wanted to produce a menu access technique where novice operation was a rehearsal of expert behavior. Essentially, we wanted novices and experts to use the same menu access technique perhaps differing only in speed of operations (much like Marking Menus [8]).

OTHER POSSIBLE SOLUTIONS

Our default solution in Maya makes use of the traditional menubar. Due to the sheer number of menu-items, we divide the menu-items into separate menubars which roughly reflect the high-level tasks of our users. Therefore we have “Modeling”, “Animation”, “Dynamics” and “Rendering” menubars. The first six menus of a menubar are common to all menubars so we refer to this subset of menus as the “Common” menu-set. These four menubars cannot be displayed simultaneously so a user can switch between menubars by selecting from a popup menu containing the four menubar names which is located directly underneath the main menubar. The menu items in this modal menubar can also be accessed with other standard GUI techniques: predefined hotkeys and user definable “drag and drop” toolbars and hotkeys.

This default solution does address some problems mentioned in the previous section. The multiple menubars increase the virtual capacity of the menubar by a factor of four. Hotkeys do provide fast access to functions in the menubars but as mentioned earlier the number of hotkeys is limited and the mappings are confusing. The problem of breaking functional groupings by distributing functions between toolbars and menubars is eliminated by simply placing all functions in the menubars and having no predefined static toolbars.

However, there are some major drawbacks to this solution. Direct menu access (not using hotkeys) is slower when a user has the extra step of switching between menubars. Also, our user base finds menubar and toolbar selection slow since selection requires having to move the cursor from their working area in the center of screen to the menubar or toolbar at the edge of the screen and back. Relative to hotkeys and in-context pop-up menus this “cursor round trip” is perceived as slow.

This solution also has unification problems. Novice and expert behaviors are radically different: novices use the menubars while experts use hotkeys and user defined toolbars. Drag and drop toolbars further erode unification, since favorite commands and unfavorite commands get placed in different spots over time breaking functional groupings.

Given the problems with this “status quo” solution we began work on alternate solutions. Some solution approaches we ruled out early in our design process. Our solution space was narrowed down dramatically by the fact that we were designing for a commercial product not a research system. For example, text command line entry was ruled out for the usual arguments that apply concerning GUI versus command line systems. Speech input was ruled out because systems support for speech isn’t ubiquitous and guaranteed like keyboard and mouse input.

Other solutions such as menu modes which present fewer menu-items to novices than to experts [3] were not suitable because they only aid novices not experts who need to deal with the complete set of menu-items and also require fast access.

Essentially we had to construct a solution out of the basic interaction techniques of GUIs: popup, pulldown menus, different mouse buttons, hotkeys, etc. None of these techniques used in the traditional sense provided a satisfactory solution. The main problem with these techniques was handling the sheer number of menu items without producing heavily nested menus (nesting menus has been shown to degrade menu selection performance [7][10]) or forcing the user to learn many different hotkey or button mappings. However, the solution we did develop is built using a combination of these GUI components.

Ultimately, we had to develop a new technique which our users would prefer over the traditional default technique. Use of the new technique would be optional (both the new technique and the traditional technique would be available in the application simultaneously), so if users elected to use the new technique instead the traditional we would consider this a success.

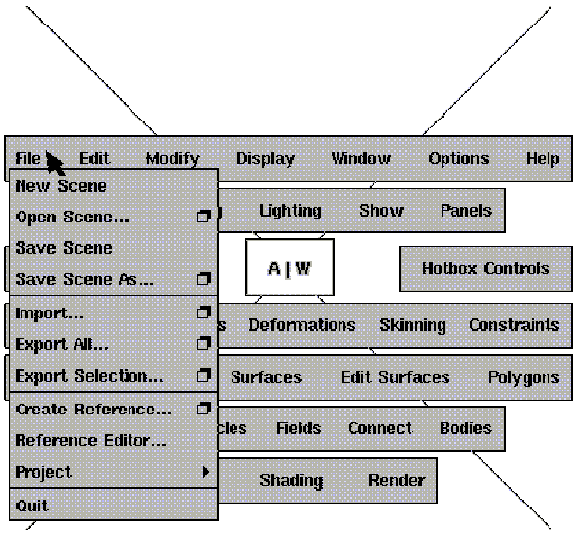

Figure 3: Hotbox widget with linear popup menus.

HOW THE HOTBOX WORKS

This section outlines how the Hotbox widget works. The next section will discuss rationale behind this design.

The HotBox works as follows. To display the Hotbox the user holds down the space-bar (with their non-dominant hand) when the cursor is in any of Maya's windows. The Hotbox instantly appears, centered at the location of the cursor. The "rows" of the Hotbox (see Figure 2) behave like traditional menubars. Individual menus can be popped down like menubar menus by moving the mouse (with the dominant hand) so the cursor is over a menu label and pressing any mouse button (Figure 3).

Each row of the Hotbox corresponds to a particular set of menus (Figure 4). The top row, is referred to as the "common" row. This corresponds to menus commonly found in most applications' main window menubar (e.g., File, Edit,...). The next row down shows the items in the menubar for the window that the cursor is currently in. Below the center row of the Hotbox are rows of menus specific to certain computer graphics tasks.

The center row's menu label "Recent Commands" displays a list of recent commands issued by the user and allows a user to repeat a command without having to relocate the command in the menus. (Figure 5). The other menu in the center row, "Hotbox Controls" allows the user to control which rows of the Hotbox are displayed. This menu is a marking menu [8]. In Figure 2, all the rows of the Hotbox are displayed. Using the marking menu a user can quickly display and hide specific rows. Figure 6 shows an example of changing the display of rows.

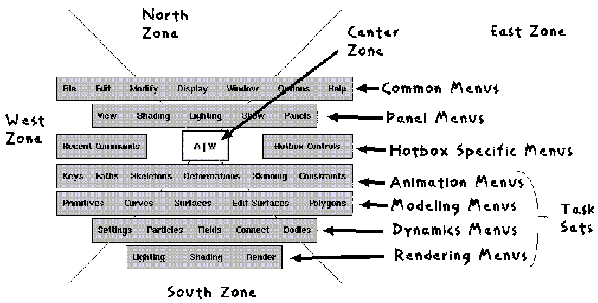

Figure 4. Hotbox rows and marking menu zones for widget.

Besides presenting the user with rows of menus the HotBox divides the entire screen into five zones (Figure 4). Each one of the these zones has a different marking menu which can be accessed simply by pressing down a mouse button when the cursor is in the zone. These marking menus are used for user defined menus.

The Hotbox remains displayed as long as the space-bar is kept pressed. This allows a user to perform a series of commands without having to re-invoke the Hotbox.

DESIGN ISSUES AND RATIONALE

In this section we present major design issues and rationale in the development of the Hotbox. We have grouped these issues and rationale into five main categories: Quantity, Speed, Unification, Graphic Design, and Interaction Design.

1 Quantity

The Hotbox project actually started off with the goal of simply increasing the number of marking menus we could make available in our applications and perhaps capturing all the menu-items in Maya in these marking menus. Traditionally, a user can access marking menus in our older product (Power Animator) by holding down both the shift and ctrl key (the "trigger key") and pressing a mouse button to popup a marking menu. Each mouse button has a different marking menu associated with it. If the user configures each menu to have eight items, this results in fast access to 24 items.

This configuration of marking menus in Power Animator is extremely popular--most expert Power Animator users make heavy use of this feature. In Maya we wanted to improve the situation:

- use the space-bar instead of the awkward "shift-control" keys.
- have the same menu for each mouse button (to avoid "wrong mouse button" errors and to support use with a tablet and stylus device instead of a mouse).
- get access to many more items.

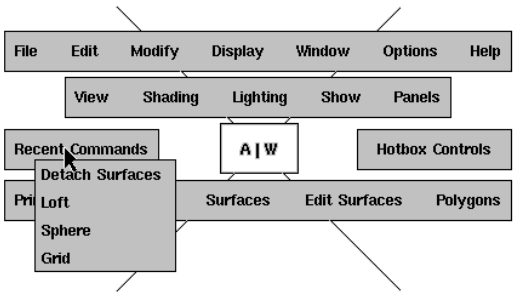

Figure 5. Recent commands history.

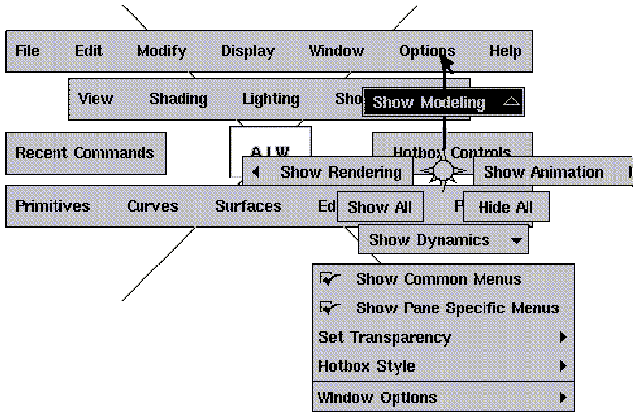

Figure 6. Marking menu used for switching rows in Hotbox.

Based on these requirement we developed the idea of using different "menu zones" to access different menus as opposed to different mouse button or trigger keys. To get a feel for this solution approach we built several working prototypes and let expert users test them. Figure 7, shows variations on the configurations of zones.

Our expert users reported that this approach allowed efficient access to menus. However, we still realized that 4 or 5 zones were not enough to capture all the menu-items in Maya without severe menu nesting. To solve this problem, we then generalized the concept of zones to have overlapping zones and developed the basic configuration of the Hotbox. With this new approach we realized we could begin considering handling the number of menu-items in Maya.

The next question was how to organize the menu-items in the Hotbox.

Menubar compatibility. In designing the Hotbox widget, we wanted to be sensitive to users who worked with the traditional GUI menubar and then wanted to transition to the Hotbox. To support learning, we designed the Hotbox command set rows to match the menu organizations in the traditional GUI menubar.

Hide/show command sets. While the original intent for the Hotbox was to house and present all of the command functions to the user, we learned early on that users did not want to see all of the command sets all of the time. Therefore, we created a marking menu (Hotbox controls) to allow the user to quickly toggle the visibility of individual rows or to specify the viewing of specific rows (which hides all other rows).

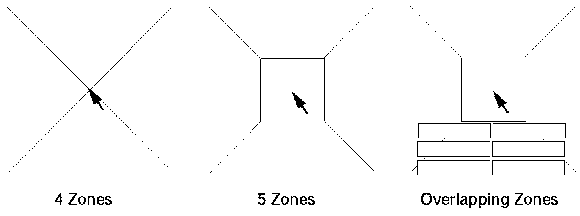

Figure 7. Early prototypes of Hotbox with marking zones.

Pane specific command set. Besides the main menubar, each window in Maya may have its own menubar. The Hotbox provides access to these menus as well. The "pane specific" row in the Hotbox changes its contents depending on the position of the cursor within different views at the time the Hotbox is invoked. This design provides context specific access to command sets which automatically change as the user changes views.

2 Speed

Large menu targets. With the layout of the Hotbox, there is a design tension between speed of access (making the menu rows tall and wide) versus the overall size of the Hotbox widget (which interferes with seeing the underlying application data). We know from Fitts' law [12] that speed of target acquisition is a function of the distance to the target from the cursor and the width of the target. In our case, the width of the target is broken up into two components: the length and height of the menu label.To provide fast access, we increased the height of our menu rows which is the true effective width of our Fitts' Law targets while not distorting the visual appearance of the Hotbox widget.

Popup at cursor or center of screen. Having chosen a popup design, we considered two strategies. First, we could pop the Hotbox widget centered around the current cursor location. This reduces the average distance between the cursor and a given menu label. Also, it guarantees that the center marking menu zone will be immediately active for invoking a marking menu. Alternatively, we considered popping up the Hotbox widget in the center of the screen. This design would provide a constant, absolute positioning of menu items as well as marking menu zones. In terms of cursor travel distances, this approach is more costly than the first approach where the menus come to the user instead of the user going to the menus.

Issuing multiple commands in a single posting. Many popup widgets are designed to dismiss themselves when a user selects an item. We designed the Hotbox widget to handle issuing multiple commands in a single posting. This provides a more efficient interaction (often saving mouse clicks and cursor travel time).

Marking zones. The 5 marking zones (North, South, East, West and Center) are designed to provide a user with quick access to a large number of their own personal custom menus. These customizeable marking menus are extremely useful for an expert user. For example, a user that very frequently invokes the "show surface normals" menu-item (which is nested 2 levels deep in the menubar menus), can place this item in one of the zone marking menus for fast access.

These zone menus can be built by a user using a GUI menu builder in Maya. Not only can they contain menu-items from Maya's menubars but a user can also write their own custom menu-item command using Maya's embedded programming language. This level of customizability is extremely useful for expert users.

Quick access is supported in two of ways. First, a menu zone is an extremely large target (almost 1/4 of the screen). This makes moving the cursor into it extremely fast. Second, the use of marking menus provides fast command access compared to traditional linear menus [8]: once in the correct zone a user can simply press down the mouse button

and “flick” the cursor in the direction of the desired menu-item.

Within each zone, a marking menu set can be defined for each of the three mouse buttons. This provides the user with the potential for 15 custom marking menu sets. While this may seem a bit excessive, we believe it is a reasonable size if users have a preference for single level menus. By default, the Maya application has five default marking menu (one in each zone). The center marking menu zone has been designed for the quickest access as it does not require any cursor travel before a user can start issuing a mark.

By its design, the Hotbox creates a hierarchy of access speed. The center zone provides the fastest access. When the space bar is pressed it can be accessed without having to move the cursor. The next level of access is the north, east, south and west zones. These zones don’t pop-up under the cursor but since they are very large they can be moved to very quickly. Finally, the menu row items are the next level of access. Like the zones, they require cursor movement but are slower to access since they are much smaller targets than the zones. Also, within the menu rows, items closer to the center are faster to access.

3 Unification

Everything in menubar. Early versions of Maya had two main widgets to house commands: the menubar and toolbox. Our users never realized that the toolbox contained moded items. Instead they would ask us why the functions were separated into two places. Given this, we wanted to provide “one-stop-shopping” for our users where the commands were organized by function not by interaction style (e.g., moded tools are in the toolbox while one-shot actions are in the menus). Placing the tools into the menus had a side benefit of using text labels instead of icons to describe a command function.

Menus under one key. To simplify the interaction model, we wanted to define a single mechanism for accessing the menus. While Maya uses the Motif toolkit, which has it’s pop-up menus under the right mouse button, many of our users found this fatiguing and wanted to use the left mouse button. We could not use modifier keys (ctrl, alt, shift) as these were already assigned to standard keys for managing selection and camera controls. Thus, we needed to find another key and we chose the space bar for its ease of access. Using a single key to access menus within the application provides a gestural unification and simplification to the overall interaction model.

Customizeable menus. Strictly speaking, our customizeable zone menus violate our design principle of unification. Like a drag and drop toolbar a user can relocate functions and this results in different novice and expert behaviors. We have two observations concerning this. First, because of the capacity of the Hotbox zones, our default zone menu set is quite large and we have observed that many expert users don’t find a need to create custom menus--they simply use the defaults and perceive these menus as the functional location for these menu-items. In this situation novice and expert behavior remains the same. Second, when a user creates fast access to some menu items by placing them in a zone menu, the basic access behavior remains the same (i.e., through the Hotbox). This is in contrast with traditional drag and drop customization where a menu-item is dragged from the menubar to a toolbar after which access is through the toolbar not the menubar.

4 Graphic Design

Visual & organizing theme. We considered a variety of layouts for the menu sets (e.g., column based, cluster based, etc.). In the end, we chose a row based approach which was easy to implement, offered a compact layout, and visually reflected the common menubar concept.

Row presentation. Presenting multiple menu rows to the user without overwhelming them was a major challenge in our design. Initially we had the rows left justified. Next we prototyped a center justification approach to reduce the travel distance to the various menus (see Figure 8). Still, graphically, this was hard to visually parse and identify command sets based on row length. We considered coloring each row separately but realized that some of our machines only have 3 colors in the overlay window in which the Hotbox widget is drawn so this solution would not work. Finally, we came up with a layout algorithm which we call “stair-step justify” which quantizes row lengths to various uniform step increments and center justifies the row. This provides visual order to the Hotbox widget (see Figure 2). In addition, we placed a border around the rows to further reinforce their menubar likeness and to reduce visual interference from the application data. Lastly, we preserved the row ordering (Common, Pane specific, Hotbox specific, Animation, Modeling, Dynamics, Rendering) and made the Hotbox specific row visually distinct to provide a visual grouping of rows. This final design is much more visually balanced while still offering the same degree of interaction efficiency as in our earlier designs.

Marking menu zones. Delimiting the five marking menu zones also provided a challenge. We quickly settled on using the cross (X) but found it difficult to determine visual rules for the length of the lines. This was specially awkward for Hotbox configurations with an even number of rows. This is truly a subtle graphical design issue which was noticeable and visually disturbing in our early designs but has been rectified in our current design (which keeps the cross length balanced above and below the Hotbox specific row, such that the cross is perfectly square).

Transparency. To reduce obscuring the underlying application data, the Hotbox widget employs transparency (see Figure 1) to allow the user to see through the widget (similar to systems like ToolGlass [1] or T3 [9]). This is especially useful as the Hotbox can be quite large when all of

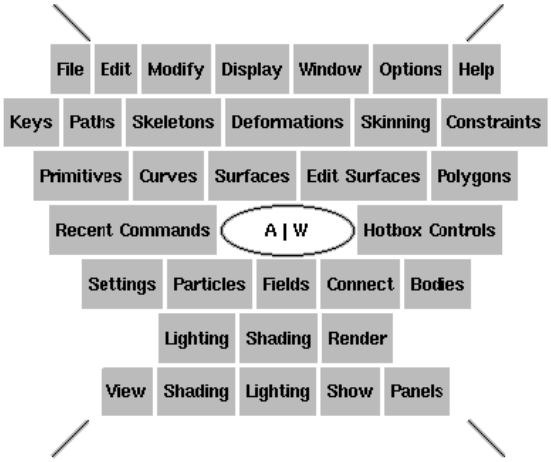

Figure 8. Early Hotbox layout with rows centered.

the rows are being displayed. The user can adjust the degree of transparency from fully opaque to clear.

Anti-alias fonts. When the Hotbox is drawn with 100% transparency (i.e., clear), a great deal of interference occurs between the textual menu labels and the underlying application data. To reduce this interference, we use anti-alias fonts [4] which surrounds each character with an "opposite" contrasting color to ensure its legibility.

No need for traditional menubars. Users also quickly realized the benefit of transitioning to the Hotbox widget in that they can hide the traditional GUI menubars (both the main menubar and the pane specific menubars) to free up more screen space for their application data. This can be a very significant saving if there are many windows displayed with a menubar in each one.

5 Interaction Design

No Hotbox warping. We do not warp the position of the Hotbox if portions of it fall off the screen. This is to produces the benefit of having the center marking menu zone always popping-up under the cursor. The cost is, of course, to have some inaccessible menu-items when popping up near the edge of the screen. We have found in practice that this isn't a major problem since users prefer to generally work near the center of the screen.

Menubar functionality. Since the Hotbox widget can be viewed as a collection of menubars, users expect the same degree of functionality as traditional menubars. Thus, we need to provide the usual functions of posting individual menus, browsing multiple menus within a single drag operation, the ability to tear-off menus, and offering roll-over help for individual menu items. Nothing in the design of the Hotbox prevents these features, however, time constraints prevented us from implementing these features in the first commercial release of the Hotbox.

HOTBOX USAGE

At the time of this writing, the Hotbox has been used on a daily basis by about 10 in-house users for 16 months, by about 100 beta customers for 13 months, and has now been shipping in the *Maya* product for 6 months, available to thousands of users. At each stage, we find approximately the same usage pattern: some users just ignore it and use the regular GUI elements instead. Some use it as part of their workflow but not to its full extent. Finally, some users use it extensively and exclusively (that is, they hide the traditional menubars, toolbars and make heavy use of the zone menus).

One useful method we have for gauging the success of a product feature is through unsolicited comments from our users, either directly sent to us, published in internet newsgroups or addressed to our customer service. As an example, Table 1 shows some representative comments found on the comp.graphics.apps.alias and comp.graphics.apps.wavefront newsgroups (notes: PA and Wavefront are our previous products, MAX is a competitor, all comments are from different contributors, misspellings are not corrected)

This data is by no means a formal proof of the efficiency of the Hotbox but it is evidence that there is strong acceptance of the hotbox from a portion of our customers.

A more formal survey of 12 of our most experienced in-house users revealed that 5 of them are intensive users, removing all menu bars from their display to gain screen real estate, 5 are frequent users (using it for about half of their menu selections but not removing all menubars from their screen) and 2 use it very rarely or never.

We should point out that the use of the hotbox is optional and users can choose to use many other standard GUI techniques in Maya instead of the Hotbox (e.g., drag and drop toolbars, user definable hotkeys, traditional menubars and popup menus). Thus, we believe that users are using the Hotbox because of some perceived benefit.

> With all the missing Tools, is anyone happy about what they got *> a paid for in Maya? Can you give me 5 good reasons to buy it?* *1) Semi-Procedural Modeling and Animation (makes PA look barbaric)* *2) FAST UI interactivity for modeling* *3) Incredible work-flow improvements over PA, Wavefront* *4) Hotbox!* *5) Hardware rendering particles.*
Does anyone who's used Maya for more than a few days ever use the regular menus? The hotbox seems so handy (if visually chaotic) I can't imagine using the traditional menus much.
I tried to do some work in PA the other day only to find that I had almost forgotten how to use it. Maya's way of doing things has taken over my brain. I was on my Mac using AfterEffects and tried to use the Hotbox, and was confused momentarily as to why it wasn't working (about 2 or 3 se onds of confusion).
As I have said I have used Maya just for 10 or 15 hrs or so, and when I come back to PA I feel really bad and I start to press Space for my hotbox menu.
"This other animator here who uses MAX was a hard sale on PA (Oh yeah, we can do that for less is basically his line) was also impressed. I heard him say, Hmmmm.....now that's nice!!! He was referring to the HotBox, general workflow issues, hardware rendered particles etc. and that sort of compliment from him is a rarity. Anyway, I can't wait to see where this is all going with MAYA.".

Table 1: Unsolicited comments from users.

CONCLUSIONS

We believe the Hotbox design produces the following benefits to a user and these benefits are responsible for its acceptance:

- Access to multiple menu bars without having to switch menubar modes.

- Fast access to up to 15 user definable marking menus. Up to 3 of these menus are available directly under the cursor after pressing the space-bar.
- Multiple commands can be issued in a single posting.
- Commands normally distributed between the toolbox, menubar, window menus and user definable marking menus are in the same spot.
- The user can free up more screen space by hiding the traditional menubars.

We believe the following features aid in learning and using the Hotbox:

- Mimicking the structure of the traditional menubar menus eases learning of the Hotbox menu rows.
- The simple and consistent access method for all menus (i.e., press the space bar and press a mouse button to pop-up a menu) is easy to learn and habituate.
- Supporting both novice and experts without requiring customization or radically different behaviors.

We also believe the Hotbox design and rationale can be applied to other domains. First it could easily be applied to other applications where the number of commands overload the traditional GUI elements. Furthermore, many of the benefits of the Hotbox are still applicable even if the application's command set isn't overly large.

Finally, we hope that other UI designers will apply some of the unique design principles and techniques used in the Hotbox (unification, a single, habituating access mechanism, large, radial zones for fast menu access, marking menus and transparency).

ACKNOWLEDGEMENTS

We gratefully thank our expert users most notably Jeff Bell, Chris Ellison and Corban Gossett for their relentless feedback. Beth Goldman, Ravin Balakrishnan and Bill Buxton also provided valuable comments on the design. We would also thank Venu Venugopal, product manager for Maya, for supporting the development and deployment of the Hotbox.

REFERENCES

1. Bier, E., A.,Stone, M., C., Fishkin, K., Buxton, W., Baudel, T., (1994) A Taxonomy of See-Through Tools. *Proceedings of the ACM CHI'94 Conference on Human Factors in Computing Systems*, 358-364.

2. Brooks, P. (1994). Adding Value to Usability Testing. in *Usability Inspection Methods*, Nielsen, J. & Mack R. (Eds). John Wiley. 255-271. see p. 262.

3. Carroll, J., M., & Carrithers, C. (1994) Training Wheels in a User Interface. *Communications of ACM*, 27, 800-806.

4. Harrison, B. & Vicente, K. (1996) An Experimental Evaluation of Transparent Menu Usage.*Proceedings of the ACM CHI'96 Conference on Human Factors in Computing Systems*, 391-398.

5. Gould, John (1988). How to Design Usable Systems. in *Handbook on Human-Computer Interaction, M. Helander (Editor), North-Holland. Elsevier. 1988.* pp. 757-789. Reprinted In *Readings in Human Computer-Interaction: Towards The Year 2000.* Baecker, R., Grudin, J. Buxton, W. & Greenberg, S. (Eds), Morgan Kaufmann, 1995. 93-121. see p. 113.

6. Jeffries, R. (1994). Usability Problems Reports: Helping Evaluators Communicate Effectively with Developers. in *Usability Inspection Methods*, Nielsen, J. & Mack R. (Eds). John Wiley. 273-294. see p. 278.

7. Kiger, J.L. (1984) The Depth/Breadth Tradeoff in the Design of Menu Driven User Interfaces. *International Journal of Man Machine Studies*, 20, 210-213.

8. Kurtenbach, G., Buxton, W. (1993) The limits of expert performance using hierarchical marking menus. *Proceedings of CHI '93 Conference on Human Factor in Computing*, 482-487.

9. Kurtenbach, G., Fitzmaurice, G., Baudel, T. & Buxton, B. (1997) The Design of a GUI Paradigm based on Tablets, Two-hands, and Transparency. *Proceedings of the ACM CHI'97 Conference on Human Factors in Computing Systems*, 35-42.

10. Landauer, T.K. & Nachbar, D.W. (1985) Selection from Alphabetic and Numeric Trees Using a Touch Screen: Breadth, Depth and Width. *Proceedings of the ACM CHI'85 Conference on Human Factors in Computing Systems*, 73-78.

11. Lewis, C. & Rieman, J. (1993). Getting to Know Users and Their Tasks. in *Task Centered User Interface Design, a practical introduction*. Reprinted In *Readings in Human Computer-Interaction: Towards The Year 2000*. Baecker, R., Grudin, J. Buxton, W. & Greenberg, S. (Eds), Morgan Kaufmann, 1995. 122-127. see p. 124, col. 2.

12. Mackenzie, I.S., & Buxton, W. (1992) Extending Fitts' Law To Two-dimensional Tasks. *Proceedings Of Acm Chi '92 Conference On Human Factors In Computing Systems*, 219-226.

13. Sears, A. & Shneiderman, B. (1994) Split menus: Effectively using selection frequency to organize menus. *ACM Transactions on Computer-Human Interaction, vol. 1, #1 (March 1994), 27-51.* also available online at ftp://ftp.cs.umd.edu/pub/papers/papers/2997/2997.ps.Z.

14. http://www.aw.sgi.com, Maya product brochure