BROWN UNIVERSITY
Department of Computer Science
Master's Project

CS-95-M6

"3D Menu: Text Menus in a 3D World"

by

Jovanna Ava Ignatowicz

3D Menu: Text Menus in a 3D World

Jovanna Ava Ignatowicz

Department of Computer Science
Brown University

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in the Department of Computer Science at Brown University.

May 1995

Professor Steve Reiss
Advisor

Introduction

In recent years, 3D computer graphics has been increasing in popularity, power and potential. New hardware and software is constantly being developed for 3D graphics both in industrial and research environments. Along with this explosion has come much debate regarding proper navigation techniques and user interfaces that interact with 3D scenes. In fact, little is known about optimal 3D user interfaces. Some preliminary work has been done, but much research remains to be done in this area.

My master's project is one implementation of a 3D user interface. 3D Menu aims to address a number of concerns dealing with user interfaces in 3D, discussed below, but primarily it is meant to be a useful and efficient three-dimensional object-oriented text menu.

Related Work

A 3D text menu is not an entirely new concept. 3D text menus have been incorporated into various applications as a means of interacting with desktop and immersive 3D environments. Although some researchers mention the textual user interfaces implemented in their specific projects, very little work has been done on general, multi-purpose desktop 3D menu toolkits. 3D menu is a step in this direction.

In immersive environments, there exist several notable 3D menus, such as University of North Carolina's 3DM [BUTTERWORTH], National University of Singapore's M-Cube and SuperCube [SERRA] and the text menus in the Virtual Wind Tunnel project at NASA Ames Research Center [ELLIS]. The latter is a 2D menu manipulated by a 6 degree-of-freedom device. Although it is simple, organized in a fashion familiar to users of 2D interfaces, and non-obtrusive (being screen-aligned and modal) it is extremely difficult to use due to the interaction device. The tracker that the application uses does not easily allow the user to bring up the menu, navigate through submenus and make a selection. Furthermore, the menus are completely flat and incongruous with the scene, which is otherwise entirely 3D.

Brown University Computer Graphics Group's Tricorder is another example of a 3D menu in an immersive environment and also the one that has had the most influence on the design of 3D Menu. It is a simple, flat menu that appears near the interaction device in the scene and allows the user to highlight and select some menu option. It serves its purpose well (simple menu selections) but has only one type of menu selection and no hierarchical structure.

In desktop environments, Xerox PARC's folding menus [CARD] stand out as an example of a useful 3D menu. The menus are flat squares that lie on the "floor" of the scene, yet remain readable. When the user activates a menu button, it flips up and lets the user make a selection. These menus are relatively unobtrusive and highly intuitive. They remain in a fixed area of the screen and are not 3D in structure, but still integrate well into the scene.

Considerations For Constructing 3D User Interfaces

3D user interfaces present many considerations, mostly involving user cognition and perception. Some of these considerations are basic user interface concerns while others are unique to 3D design. Ideally, a 3D user interface should address all of these concerns to some extent. Many do not, simply because 3D user interface design is still in its early stages.

Special problems abound when one makes the transition from 2D to 3D. For instance, input devices may change. Since picking ease is essential to an application's success, the 3D interface must be easy to activate and manipulate. This is often a simpler problem in desktop environments where the mouse is typically the input device of choice. In a 3D application, there is also a number of 3D objects that can change position in the scene. An integrated 3D user interface may or may not be a 3D object, but if it is, it must avoid overly obstructing the scene, and keep from being obstructed itself.

A 3D application must also avoid visual clutter and make proper design decisions regarding the placement and manipulation of the user interface. If the user interface components

are first-class 3D objects, they need to have proper lighting, shading and shape. They also need to take rendering speed into account and stay simple to remain real-time.

As for traditional user interface concerns, 3D user interfaces should embrace the attributes of their 2D counterparts that carry over into 3D. For instance, they should have rich attributes and predictive highlighting. One of 3D Menu's main goals is to incorporate as much 2D interface design into 3D as needed. This also enhances the user's familiarity with the interface and its functionality, and makes the application more coherent.

3D Menu- General Design

The general design of 3D Menu is partly influenced by previous work. Certain advantageous aspects of previously implemented 3D menus are embodied in its design, but it is mostly a direct application of general user interface techniques to the task at hand: a simple, easy to use, extendable 3D text menu.

First and foremost, 3D Menu is designed to be easy to use. It is very similar in look and feel to traditional 2D menus (i.e. Motif.) A user is automatically familiar with its structure, shape and behavior. It has predictive highlighting, Motif-like icons and is activated and manipulated using the mouse.

3D Menu is considerably rich in attributes compared with other 3D menus. It supports basic hierarchical menus consisting of string, toggle, radio, submenu and dialog selections. To some extent, it can fully replace some simple 2D menus made up of these types of buttons. If

the user wishes to further extend the user interface, the dialog selection button can be used to bring up a standard Motif menu. If the programmer wishes to extend the package, he or she can use 3D Menu components to build new menus and/or buttons.

As a 3D object, the menu avoids obstruction by only appearing upon activation, disappearing upon de-activation, staying screen-aligned and very close to the view plane. It does, however, obstruct any application objects that lie behind it in the view plane. Since it is modal and drawn only when the user invokes and interacts with it, this is actually inconsequential. The objects need not be visible when a menu is active. Its position is determined by the position of the mouse cursor at that time the mouse button is pressed.

3D Menu has lighting and shading like any other 3D object and is composed of few, but attractive polygonal objects for fast rendering. Because it is screen-aligned, it uses hardware-supported annotation text that does not add any polygons to the scene.

Finally, 3D Menu differs from traditional 2D menus in that it is implemented in the same 3D environment as the application. Therefore, it provides a tighter semantic coupling between the application and the user interface, and places the menus in the 3D scene. Since it is implemented in Trim Light, a graphical package written in C++, it takes full advantage of Trim Light's object-oriented structure. Each menu button is an object and the collection of all the menu buttons in a menu make up the menu object. The menu object can then be linked to either an application object or the scene space (background) and is invoked when the user clicks on that object.

3D Menu- Specific Design

Look-And-Feel

Although 3D Menu is a 3D user interface, it greatly resembles Motif and traditional 2D menus in the way that it looks and behaves. 3D Menu is a 3D object, yet unlike application objects it remains screen-aligned and immobile while visible. Five types of buttons make up the 3D Menu: selection, toggle, radio, submenu and dialog buttons (figure 1.)

Selection

This button has some action associated with it. When the user releases the mouse while it is on a selection button, that action will be performed and the menu will disappear.

Dialog

The dialog menu button will bring up some dialog box when it is activated. The dialog box will be a Motif dialog already supported by Trim Light. Its purpose is to allow the user to extend their user interface by adding Motif components if necessary.

Toggle

The toggle button represents a boolean value. When it is selected, the toggle is turned on if it was off and off if it was on. If an optional function taking a boolean value is passed into the constructor, it is called with the new value. A toggle that is on is represented by a square that is depressed in the menu button and an off toggle is represented as a raised square. Since menu but-

tons are objects, the toggle button object contains its own state information and can be queried.

Radio

The radio menu button represents a boolean value, like the toggle, but also belongs to a group of radio buttons in which exactly one radio button is on at any time. A radio button that is on is represented by a depressed diamond and an off radio button is represented by a raised one. Like the toggle, the radio button can be queried regarding its state and can be passed a function to call with a boolean value when the state changes.

Submenu

The submenu button will bring up another menu next to it when it is activated. Activation only occurs when the mouse is located near the region of the arrow. Once the submenu is visible, the user can make selections from it. The string on the button typically ends with "..." in traditional 2D toolkits. 3D Menu adheres to this tradition and adds the three dots if they are not present in the string passed into the constructor of the submenu.

Interaction

Menu objects are associated with some 3D object in the scene, or the scene's background. When the mouse button overloaded for menu invocation (specified by the user upon creation of the menu viewer) is pressed on this object, the menu appears with the middle of the topmost button directly under the mouse. In most, if not all, 2D user interface toolkits, menus are drawn within the boundaries of the screen. This idea was abandoned in 3D menu as an example of

3D objects behaving too much like their 2D counterparts.

Once the mouse button is down, the menu is drawn and the mouse lies inside its boundaries. As long as the mouse button is down, the user can still highlight different menu selections. A menu button is highlighted if the cursor is in its boundary. If the user moves the cursor outside the menu and releases the mouse button, the menu will disappear and no selection will be made. If the user releases the mouse button while a menu button is highlighted, that menu button will be selected.

Hierarchical Menus

3D Menu supports a complete hierarchical menu structure. A menu is a collection of buttons, any of which can be a submenu button which is associated with some other menu. The submenu is drawn when the mouse enters the region of the arrow on a submenu button. When the submenu is activated, it is drawn with the middle of its top selection directly under the mouse cursor, just like its parent. If the user leaves its boundaries, the submenu is undrawn and no selections are made from it. If the submenu contains another submenu, the next submenu is activated and drawn in the exact same way. Since menus are first-class 3D objects and drawn physically on top of their parent menus, submenus appear larger than their parent menus because the are drawn closer to the user.

3D Menu tracks the mouse through these interactions in the following way: If the mouse leaves the boundaries of a menu (whether the root menu or any submenu) 3D Menu undraws all submenus. If the mouse leaves a submenu but enters its parent, only the submenu is

undrawn. Predictive highlighting applies to all menu buttons, including submenus, and releasing the mouse in any submenu will select the highlighted button.

Lighting

3D Menu has a separate light it creates to illuminate the menu when it is visible. This is particularly helpful in poorly-lit scenes where the menu is often drawn in shadow. It also aids in maintaining a more consistent view of the menu, since it is after all a user interface. By default, the light is a directional light that shines from the camera to the menu's position. The user can also create and pass in a custom light to the scene, perhaps making it a point light, a colored light or a stronger/weaker light.

Programmer's Interface

Once a menu is created, it is a 3D object that is manipulated with the mouse and has well-defined behavior. At the same time, there exists a programmer's interface, as with any other user interface toolkit, that allows the programmer to dynamically change the menu constructs he/ she has created. More detail is provided in the documentation.

All Buttons:

All buttons can be activated or de-activated by a direct call, and their state can be queried at any time. They can also be highlighted and unhighlited as well as manually selected. Although they all behave the same when highlighted (change to red) and unhighlighted (change back to their original color) the select method is different for every button. It behaves as if the user

selected the menu button with the mouse. Every button also has a name that can be queried but not changed. The name corresponds to the text string on the button.

Selection Buttons & Dialog Buttons:

Since these are the simplest of all the buttons, they have no extra methods aside from the basic activation, highlighting, name, and selection methods.

Toggle Buttons:

Toggle buttons have a value function that returns a boolean value depending on the toggle's state (whether it is on or off.) They also can be queried as to which radio group they belong to if they are radio toggles. If they are independent boolean toggles, the function returns a null value.

There is no separate radio button class. Boolean toggles become radio toggles when they are added to a radio group and change their geometry accordingly. When they are removed from a radio group, they revert to being boolean toggles.

Submenu Button:

The programmer can call a method to get the submenu's parent menu. Besides that, the submenu has only the other basic menu button methods.

Radio Groups:

3D Menu has radio groups to define sets of boolean toggles where only one can be on at any time. The programmer must create a radio group for each radio set that is needed. Once a group is created, toggle buttons can be added and subtracted from the group. The group makes

sure that only one is on by the following protocol: when a toggle is added, if it is on and there are already toggles in the group, it is turned off. If the added toggle is off and being added to an empty group, it is turned on. The programmer can also ask the group whether a particular toggle is in the group.

Menus:

A menu is created separately from its buttons. Once it is created, buttons can be added and subtracted from it. Unlike the radio group, adding and subtracting buttons affects the geometry of the menu since the menu buttons might be drawn in different orders. A menu button can be added to the end of the list of the menu's buttons and therefore be drawn as the bottom-most button or added before or after some other particular button.

Menus have several query methods. They can be asked whether or not they contain some button. They can also be asked for their parent menu and level. The menu drawn when the button is depressed on an object (often called the root menu) has a level number of 0. Any of its submenus would have a level number of 1 and so forth. In other words, the level of a menu is the level of its parent menu + 1.

Menu groups:

Part of 3D Menu's uniqueness comes from the fact that a 3D menu can be associated with a 3D object inside the scene. Object-menu pairs must be registered with the 3D menu as menu groups. The programmer is responsible for the creation of a menu group for each pair and must register it with the menu view (discussed below.) A menu that is to be drawn when the user

presses the mouse button on the background is registered with a null value as the scene object. If some pair is registered with an object that already has a menu associated with it, the new menu will replace the previous one. Menu groups can be added and removed from the menu viewer at any time and the components of each menu group can also be queried.

The View:

3D Menu requires special mouse event handling. To make this possible, it places certain constraints on the programmer regarding the viewer. Once the programmer creates a display, timer and camera for their Trim Light application, a special menu viewer must be created so that menus can be used. This viewer is passed the display, timer and camera into its constructor and creates a special-purpose viewer. The programmer can query the menu viewer for a viewer object and treat it as an original Trim Light viewer, but this level of indirection cannot be avoided. The menu viewer is needed for the registration of menu groups.

Conclusion

In many ways, 3D Menu is extremely well-suited to its purpose. It is simple, easy to use and nicely integrated into the 3D scene. It exhibits the familiarity of traditional 2D user interfaces while taking full advantage of its object-oriented 3D nature. 3D Menu is a first-class 3D object with proper lighting, shading and fast rendering speed. Although 3D Menu is a step in the direction of a standardized 3D menu toolkit, much more work still needs to be done in the area of textual user interfaces in 3D.

I wish to thank all who have helped me in the design and implementation of this project. Special thanks to Bob Zeleznik, Ken Herndon, Dan Robbins and Professor John Hughes of the Brown University Computer Graphics Group, and most of all, my advisor Professor Steve Reiss.

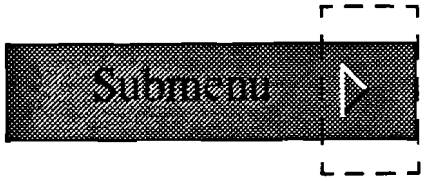
figure 1

Selection Button

Dialog Button

Toggle Button

Radio Button



Submenu Button - The vertical dotted line indicates the region where the mouse needs to be in order to activate the submenu.

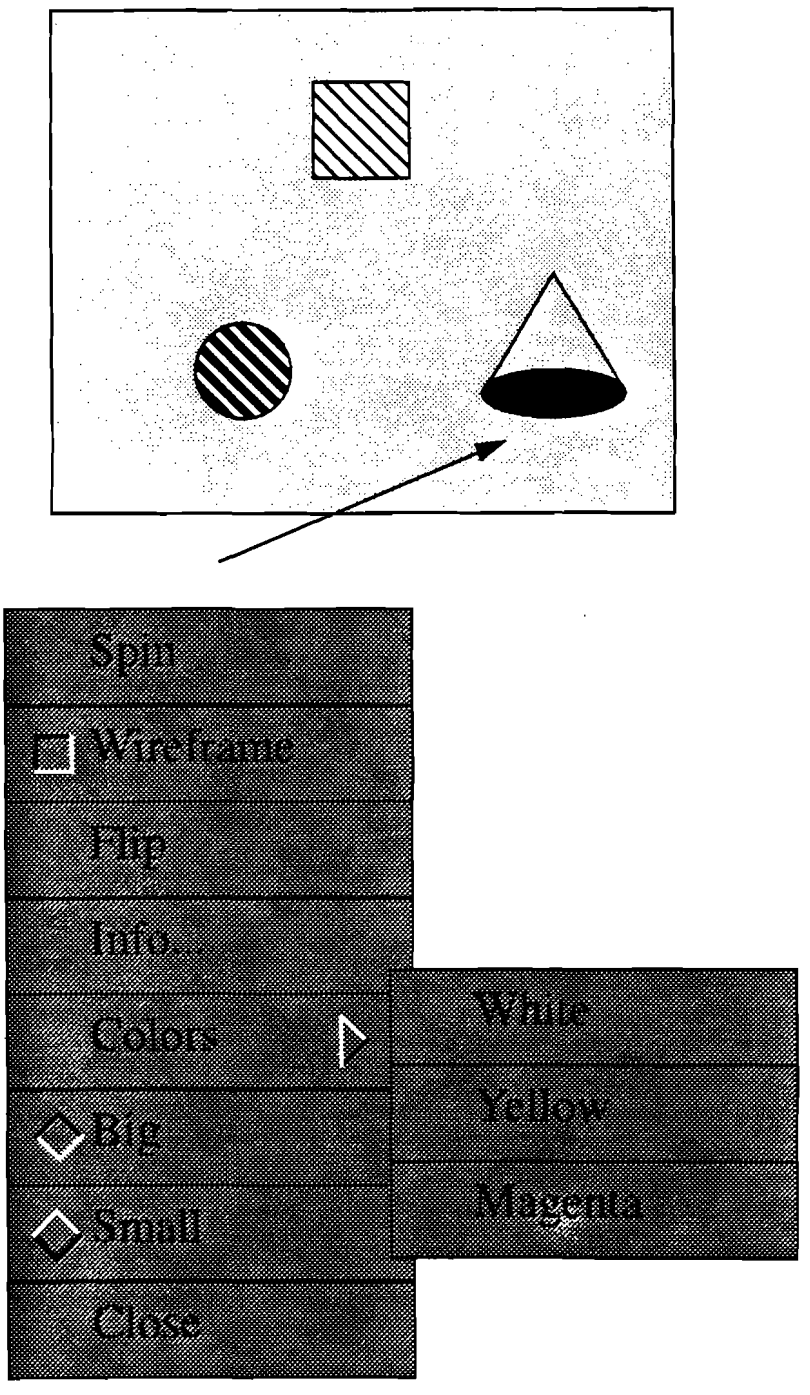


figure 2

Shown here is a sample menu. It appears when the user selects the cone with the menu-overloaded mouse button. The sphere, cube and/or background might have the same or different menu assigned to them (with the same or different callbacks) or no menu at all.

Appendix : User's Guide

3D Menu is a package that allows programmers to create 3D menus for their Trim Light applications. This document is meant to familiarize the programmer with the menu classes and their usage.

Structure: Files and Classes

MENUgrp.H

This file contains only one class: `MENUgrp`. It binds a scene's geometric object with a 3D menu (`MENUmenu`) so that the viewer knows which menu to draw when that object is selected with the "menu button" which is passed into the constructor of the menu.

```
MENUgrp(char *, GEOobj *, MENUmenu *)
```

The programmer must pass in a name for the group (need not be unique), a pointer to a geometric object from the scene, and a pointer to a menu. If the menu pointer is 0, the group is an empty group and has no effect on the program, even when passed into the viewer. If the object pointer is 0, it is assumed the menu should be drawn when the user selects the background with the menu mouse button. If the geometric object (or background) already has a menu associated with it, that menu is overwritten with the new menu.

```
~MENUgrp()
```

The destructor does not destroy the object or the menu since this class represents a relationship between the two, not necessarily a container for them.

```
MENUmenu *menu()
GEOobj *obj()
```

These two methods are accessor methods that return pointers to the objects the menu group contains.

MENU.H & MENU.C

These files contain the bulk of the 3D menu code- all the buttons and graphics. Below, each of the buttons, their interfaces and menu and radio construction is explained. The high level calls correspond to the calls a programmer will use. The low level calls are used by the rest of the

classes and might only be used when the programmer wants to do some unusual, lower level things.

MENUbutton

All buttons are derived from this class. It is not an abstract class and can be instantiated, but the resulting button is a plain button with a text string and no selection function.

High Level:

```
MENUbutton(char *)
```

The name is the string to be written on the button. The button also creates two geometric objects that it contains- the frame and the text string.

```
virtual ~MENUbutton()
```

The destructor destroys the frame and the text string.

```
char *name()
```

This function returns the button's name (text string.)

```
int active()
void active(int)
```

These functions allow the programmer to change or query the button's active state. If a button is active, it is selectable (with the mouse or through code.) If it is not active, it is grayed out and unselectable.

```
virtual void select()
```

This function calls the button's select method. In the case of this particular, generic button, it does nothing since there is no select function.

```
virtual void highlight()
virtual void unhighlight()
```

These function manually highlight and unhighlight the button (if it is active). A highlighted button is red, an unhighlighted button is its natural color.

Low Level:

```
double width()
```

This call returns the width of the buttons in the menu the button belongs to. It is used

to determine a frame's width so that all the buttons in a certain menu will be the same size.

```
virtual MENUmenu *motion(COBJcam &, COBJpt &, COBJvec &,
                         COBJpt &)
```

This function is used only by the submenu button. It is explained under the submenu button's interface further on in this document.

```
virtual OBJobj *mouse_select(COBJcam &, COBJpt &, COBJvec &,
                             COBJpt &)
virtual OBJobj *mouse_unselect()
```

These two function belong to every button so that no matter where the mouse selects or unselects, only the frame will get the call. That is, the button, text string and any icons on the button pass the information to the frame which highlights or unhighlights itself. Nothing happens if the button is not active.

MENUselection

The selection button (inherited from `MENUbutton`) is almost identical to the `MENUbutton` except for the fact that it can call some function when it is selected.

```
MENUselection(char *, void (*selectionfunc)())
```

The constructor takes a string that is to be the button's name and a function to call when it is selected or nil if there is to be no selection function.

```
virtual ~MENUselection()
```

The destructor simply destroys the class.

```
void select()
```

This calls the selection function if there is one and the button is active.

MENUdialog

This button is meant to bring up some dialog box when it is selected. The user passes it a function to call that will bring up the box.

```
MENUdialog(char *, void (*selectionfunc)())
```

The constructor takes the name and the selection function. Because dialog button text strings traditionally end with "...", this suffix is added for the user.

```
virtual ~MENUdialog()
```

The destructor simply destroys the class.

```
void select()
```

This calls the selection function if there is one and the button is active.

MENUsubmenu

This button has an arrow icon drawn on it and draws a submenu when it is activated.

```
MENUsubmenu(char *, MENUmenu *)
```

The submenu constructor takes the name of the button and a pointer to the menu that is to be drawn when the submenu is activated. Passing a nil pointer as the menu will result in a submenu button that does not draw any submenu when selected with the mouse.

```
virtual ~MENUsubmenu()
```

The submenu destructor destroys the arrow icon on its button and its submenu.

```
void select()
```

Since there is no selection function, nothing happens. The submenu is drawn when the mouse is near the arrow icon (through the motion function described below) and not the selection function.

```
MENUmenu *parent_menu()
```

This funtion returns a pointer to the menu the submenu button belongs to (not the submenu itself.)

```
virtual MENUmenu *motion(COBJcam &, COBJpt &, COBJ &, COBJpt &)
```

This funtion is called by the `MENUmouse_func` when the mouse moves inside the submenu button and the button is active. If the mouse is near the arrow, the submenu is returned (and therefore drawn.).

MENUtoggle

This button is a standard toggle. It has an on/off state, a foreground color for its icon and can be turned into a radio toggle by adding it to a radio group (described later in this document.)

```
MENUtoggle(char *, int, void (*selectionfunc)(int)=0,
           OBJvec *c = OBJvec(1,0,0)
```

The toggle button constructor takes the name of the button, its initial value (0 or 1), an optional function to call when the toggle changes value and an optional color for the toggle icon. The selection function should be a function that takes an integer, since the toggle's state will always be 0 or 1. If no color is passed in, 3D Menu uses red as the default color for toggle icons.

```
virtual ~MENUtoggle()
```

The toggle destructor deletes the toggle icons it contains.

```
void foreground_color(OBJvec)
OBJvec foreground_color()
```

Since the toggle icon can be any color (default is red) these accessor methods query and update that color. If the toggle button is really a radio button, changing the color will have no effect as the radio group's color overrides any individual toggle's color.

```
int value()
```

This function returns a 0 if the toggle is off and 1 if the toggle is on.

```
void select()
```

This function will toggle the value of the button and call the selection function with the new value (if there a selection function and the button is active.) If the toggle belongs to a radio group and is already on, nothing happens. If it is off, it is turned on and all the other toggles in the radio group are turned off.

MENUradioGroup

To convert an ordinary toggle to a radio toggle, the programmer must create a radio group and add toggles to it. Once toggles are added, they belong to the radio set, change their icon to a radio toggle icon and color it using the color associated with the radio group.

```
MENUradioGroup()
```

The radio group constructor simply creates an empty radio group and initializes the toggle foreground color ro red.

```
virtual ~MENUradioGroup()
```

The radio group destructor simply deletes the group class, but none of the toggles. Since they are no longer radio toggles, they change their geometry to be regular toggles, but keep their value and color.

```
void foreground_color(OBJvec)
```

```
OBJvec foreground_color()
```

The above pair of functions change the icon color of the toggles (present and future) in the radio group. The default is red.

```
MENUradioGroup& operator+=(MENUtoggle *)
```

This function adds toggles to the radio group (they become radio toggles and get the radio group foreground color.)

```
MENUradioGroup& operator-=(MENUtoggle *)
```

This function deletes a toggle from the radio group (if it is there.) Once the toggle is removed, it turns back to a regular toggle, but keeps the last color it had until the user changes it.

MENUview.C & MENUview.H

The programmer does not need to know about the classes `MENUobj_view`, `MENUdevice` and `MENUmouse_func`. They are created and used by `MENUview`, which is an essential part of 3D Menu.

MENUview

```
MENUview(OBJdisplay *, GEOtimer *, OBJcam *,
         int, GEOlight *menu_light=0)
```

The view is created with the display, the timer, the camera, the button 3D Menu will override to bring up menus and an optional light that is drawn when a menu is drawn. If a light is not passed into the constructor, or a null value is passed, 3D Menu creates a directional light of .5 intensity. Default and user created lights all are translated to the camera's position and point at the active menu.

```
~MENUview()
```

The destructor will clean up all classes it created, but leave the display, the timer, the camera and the menu light untouched (unless the menu light is the default one in which case it is also destroyed.)

```
MENUview& operator+=(MENUgrp *o)
```

This function adds a menu group to the viewer so that the viewer can find a selected object's menu and draw it.

```
MENUview& operator-=(MENUgrp *)
```

This function removes a group from the view (if the group is found in the view.)

```
OBJview *view()
```

This function returns the viewer that will be used in the program. In a typical Trim Light program, the user creates an `OBJview` and adds it to the timer. With 3D menu, the user must create a `MENUview` and then get the `OBJview` from it. Below is an example of how this is done:

```
MENUview mview(&the_display, &the_timer, &the camera, 1);

mview += &SomeMenuGroup;
mview += &SomeOtherMenuGroup;

OBJview *oview = mview.view();

timer += oview;
```

Code Files

Below is a listing of the files necessary for 3D menu to run. Cuurently, they are all in /pro/uga/cmd/jai/src.

Headers

MENU.H
MENUgrp.H
MENUview.H
defs.H

C++ files

MENU.C
MENUview.C
main.C

TRIP files

arrow,trip
arrow_side.trip
border.trip
border2.trip
color_plate.trip
frame.trip

Example

Below is the code needed to build the menu drawn in figure 2:

```
main(int argc, char **argv) {

    OBJdisplay disp(argc, argv);
    GEOtimer   tmr (&disp, 25);
    OBJcam     cam;
    // don't create an OBJview

    GEOobj cube(POPcube(0), "my_cube");
    GEOobj cone(POPcone(20, FALSE), "my_cone");
    GEOobj sphere(POPsphere(2, FALSE), "my_cone");

    // create the lowest level first- the color submenu

    MENUselection WhiteButton("White",ChangeToWhite);
    MENUselection YellowButton("Yellow", ChangeToYellow);
    MENUselection MagentaButton("Magenta", ChangeToMagenta);

    MENUmenu ColorMenu;
    ColorMenu += &WhiteButton;
    ColorMenu += &YellowButton;
    ColorMenu += &MagentaButton;

    // create each of the buttons in the next level (now topmost)

    MENUselection SpinButton("Spin", SpinObject);
    MENUtoggle    WireframeButton("Wireframe",0);
    MENUselection FlipButton("Flip",FlipObject);
    MENUdialog    InfoButton("Info",BringUpMenu);
    MENUsubmenu   ColorSubmenu("Colors", &ColorMenu);
    MENUtoggle    BigButton("Big",1);
    MENUtoggle    SmallButton("Small",0);
    MENUselection CloseButton("Close", CloseApp);

    // create the main menu and add the buttons to it

    MENUmenu MainMenu;
```

```
    MainMenu += &SpinButton;
    MainMenu += &WireframeButton;
    MainMenu += &FlipButton;
    MainMenu += &InfoButton;
    MainMenu += &ColorSubmenu;
    MainMenu += &BigButton;
    MainMenu += &SmallButton;
    MainMenu += &CloseButton;

    // create the radio group

    MENUradioGroup RadioGroup;
    RadioGroup += &BigButton;
    RadioGroup += &SmallButton;

    // create the menu group

    MENUgrp ConeGroup("ConeGroup", &cone, &MainMenu);

    // create the view and add above group

    MENUview mview(&disp, &tmr, &cam, 1);
    mview += ConeGroup;

    OBJview *the_viewer = mview.view();

    // add all the objects to the timer

    tmr += the_viewer;
    tmr += &cube;
    tmr += &cone;
    tmr += &sphere;

    EVENTevent_loop(disp.event_hdl());

}
```

REFERENCES

[Butterworth, 1992] J. Butterworth, A. Davidson, S. Hench, T. Olano, "3DM: A Three-Dimensional Modeler Using a Head-Mounted Display", Proceedings of the 1992 Symposium on Interactive 3D Graphics, 1992.

[Card, 1991] S. Card, J. Mackinlay, G. Robertson, "The Information Visualizer, An Information Workspace", Human Factors in Computing Systems, Proceedings of ACM SIGCHI 1991.

[Card, 1991] S. Card, J. Mackinlay, G. Robertson, "The Perspective Wall: Detail and Context Smoothly Integrated", Human Factors in Computing Systems, Proceedings of ACM SIGCHI 1991.

[Card, 1991] S. Card, J. Mackinlay, G. Robertson, "Cone Trees: Animated 3D Visualizations of Hierarchical Information", Human Factors in Computing Systems, Proceedings of ACM SIGCHI 1991.

[Ellis, 1993] S. Ellis, R. Jacoby, "Using Virtual Menus in a Virtual Environ ment", Course Notes 43- Implementing Virtual Reality, SIGGRAPH 1993.

[Serra, 1994] L. Serra, J. Waterworth, "VR Management Tools: Beyond Spatial Presence", Human Factors in Computing Systems, Proceedings of ACM SIGCHI 1994.