

Copy **Reshape** example program, make & run it (change window size with arrow keys).

Notice that the viewport doesn't change.

Examine source code: `keyboard.cpp` → `glutReshapeWindow()`  
`init.cpp` → `glutReshapeFunc`  
`view.cpp` → `reshape`

In `reshape()`

```
glViewport (VPL, VPB, VPW, VPH);

glMatrixMode (GL_PROJECTION); // manipulate the PROJECTION
glLoadIdentity ();           // matrix stack
gluOrtho2D (WL, WR, WB, WT);

glMatrixMode (GL_MODELVIEW); // manipulate the MODELING/VIEWING
glLoadIdentity ();           // matrix stack
```

Modify `reshape()`: `glViewport (VPL, VPB, ww, wh)`, make & run ...

Notice that the viewport now changes, but the objects drawn don't retain their shape.

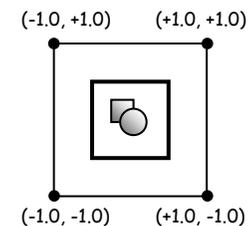
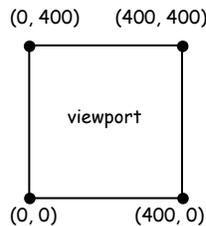
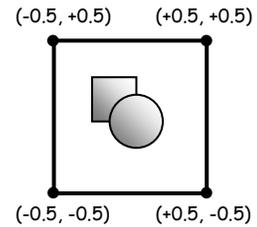
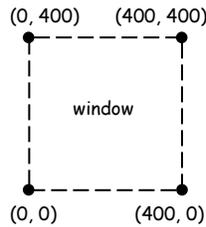
Aspect ratio - relates width to height:  $aspect = width / height$

If the viewport and projection aspect ratios are the same, geometry is rendered without distortion.

```
glutInitWindowSize( 400, 400 );
void draw( void )
{
    glBegin(GL_LINE_LOOP);
    glVertex2f( -0.5, -0.5 );
    glVertex2f( +0.5, -0.5 );
    glVertex2f( +0.5, +0.5 );
    glVertex2f( -0.5, +0.5 );
    /* draw a rectangle and a circle */
    glEnd();
}
```

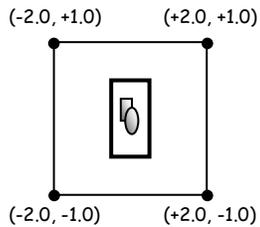
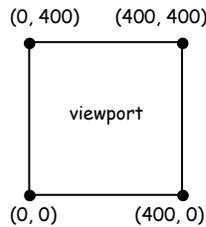
```
glViewport( 0, 0, 400, 400 );
gluOrtho2D( -1.0, +1.0, -1.0, +1.0 );
```

aspect ratios:  
 viewport: 400/400 = 1.0  
 ortho window: 2.0/2.0 = 1.0



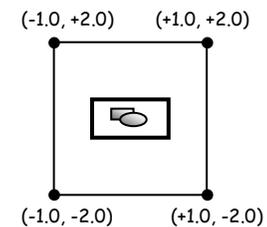
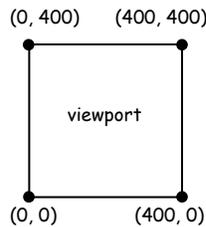
```
glViewport( 0, 0, 400, 400 );
gluOrtho2D( -2.0, +2.0, -1.0, +1.0 );
```

aspect ratios:  
 viewport: 400/400 = 1.0  
 ortho window: 4.0/2.0 = 2.0



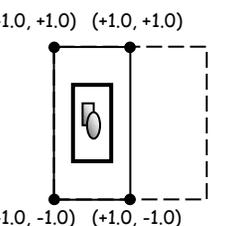
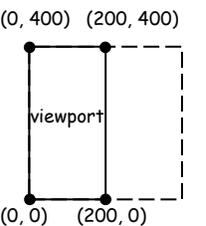
```
glViewport(0, 0, 400, 400 );
gluOrtho2D( -1.0, +1.0, -2.0, +2.0 );
```

aspect ratios:  
 viewport: 400/400 = 1.0  
 ortho window: 2.0/4.0 = 0.5



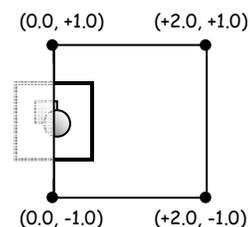
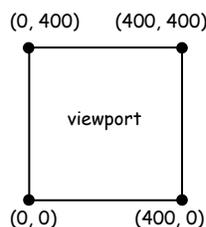
```
glViewport(0, 0, 200, 400 );
gluOrtho2D( -1.0, +1.0, -1.0, +1.0 );
```

aspect ratios:  
 viewport: 200/400 = 0.5  
 ortho window: 2.0/2.0 = 1.0

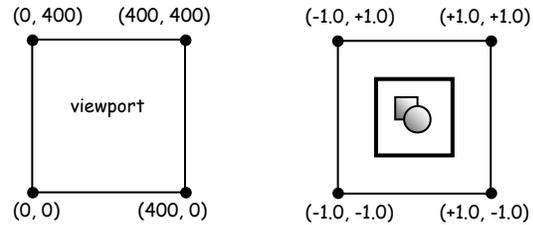


```
glViewport(0, 0, 400, 400 );
gluOrtho2D( 0.0, +2.0, -1.0, +1.0 );
```

aspect ratios:  
 viewport: 400/400 = 1.0  
 ortho window: 2.0/2.0 = 1.0



```
glutInitWindowSize( 400, 400 );
glViewport( 0, 0, 400, 400 );
gluOrtho2D( -1.0, +1.0, -1.0, +1.0 );
```

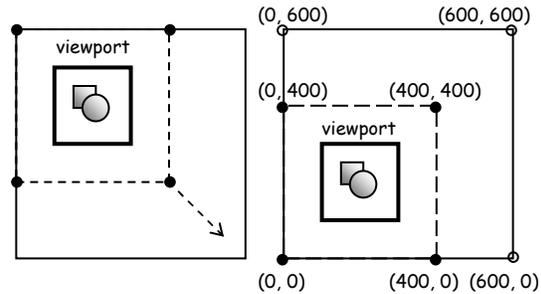


Window is resized ...

... but the viewport is still the same:

```
glViewport( 0, 0, 400, 400 );
```

A window may be resized manually or programmatically by calling function:



```
void glutReshapeWindow( int new_width, int new_height );
```

How do we resize the scene geometry when the window size changes?

### GLUT reshape callback

```
void glutReshapeFunc( void (*func)(int w, int h) );
```

Registers with GLUT, for the current window, the named function to be called when the window is resized (a reshape callback is also triggered immediately before a window's first display callback after a window is created).

Parameters **w** and **h** contain the new window size in pixels.

Function prototype:

```
void reshape( int w, int h );
```

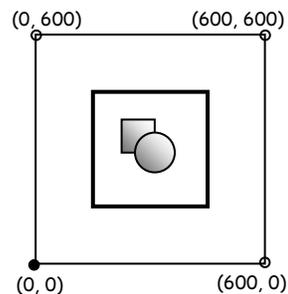
Function definition example:

```
void reshape( int w, int h )
{
    glViewport( 0, 0, w, h );
}
```

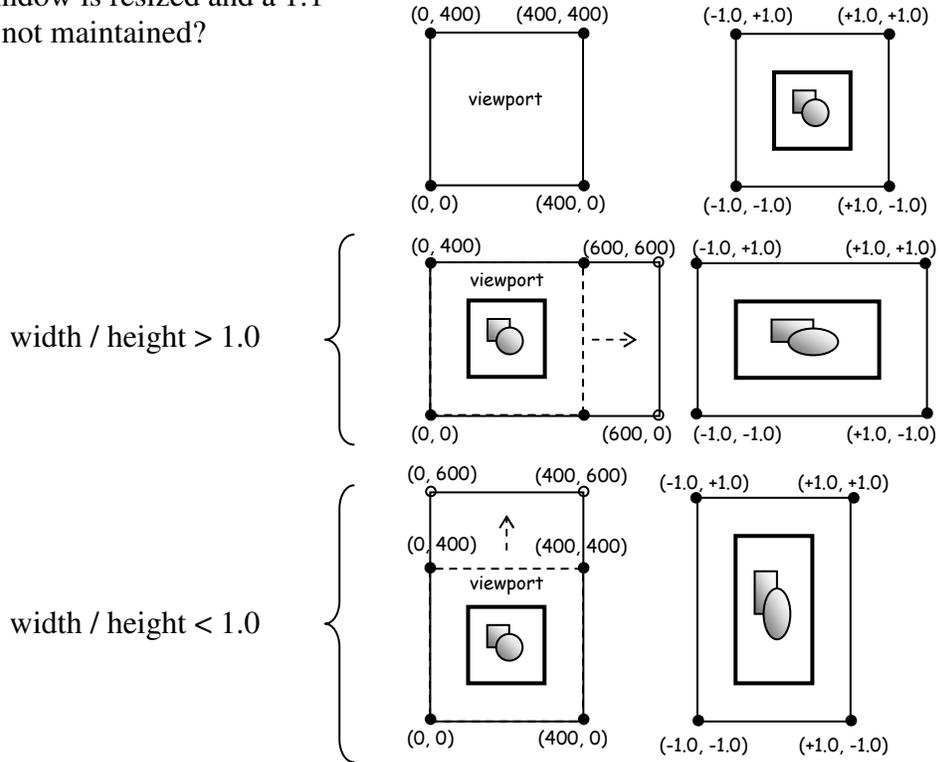
Register with GLUT:

```
glutReshapeFunc( reshape );
```

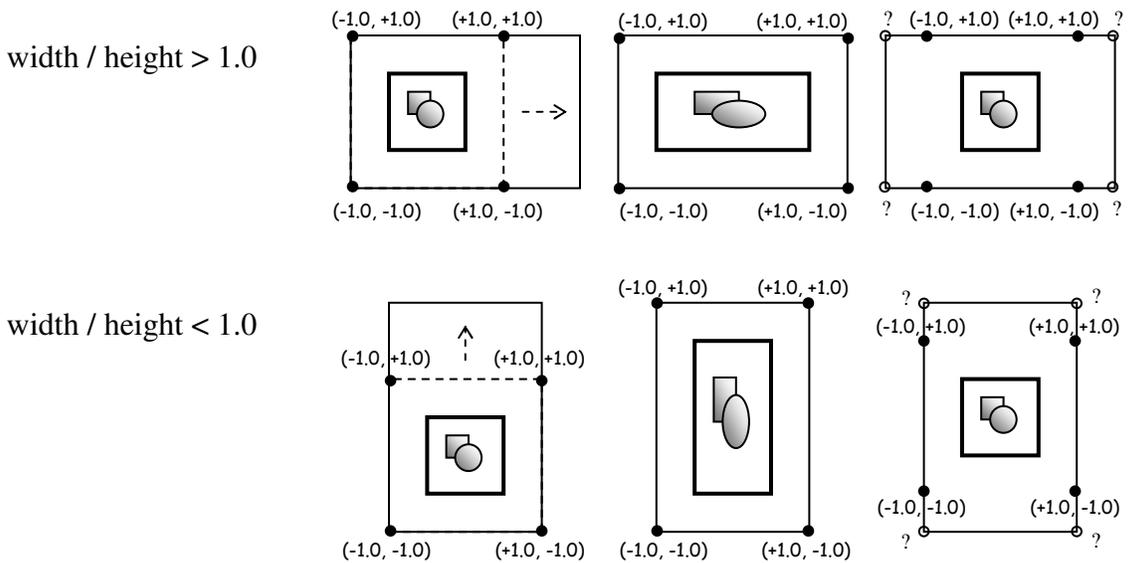
Note how the reshape callback modifies the viewport size to match the new window size.



What if the window is resized and a 1:1 aspect ratio is not maintained?



To keep a 1:1 aspect ratio we must adjust our projection:



(Walk through the **AspectRatio** program)

We'll adjust the aspect ratio in the `reshape` callback:

```
void reshape( int w, int h )
{
    if( h == 0 ) h = 1; // prevent divide by 0
    float ar = (float)w/h;

    glViewport( 0, 0, w, h );

    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();

    if( w > h )
        /* stretch left and right */
        gluOrtho2D( WL * ar, WR * ar,      WB, WT );
    else
        /* stretch bottom and top */
        gluOrtho2D( WL, WR,      WB / ar, WT / ar );

    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
}
```

Recall the steps involved in transforming world coordinates to coordinates in the viewport (viewport mapping):

1. Translate the world coordinate window to the world coordinate origin:

$$T_1 = T(-left, -bottom)$$

2. Scale to the size of the viewport:

$$S\left(\frac{vpw}{right-left}, \frac{vph}{top-bottom}\right)$$

3. Translate to the viewport origin:

$$T_2 = T(vpl, vpb)$$

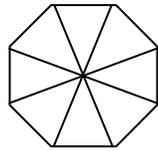
This can be expressed as a composite transformation matrix:  $M = T_2 \cdot S \cdot T_1$

The OpenGL/GLUT calls:

```
glViewport( vpl, vpb, vpw, vph )
gluOrtho2D( left, right, bottom, top )
```

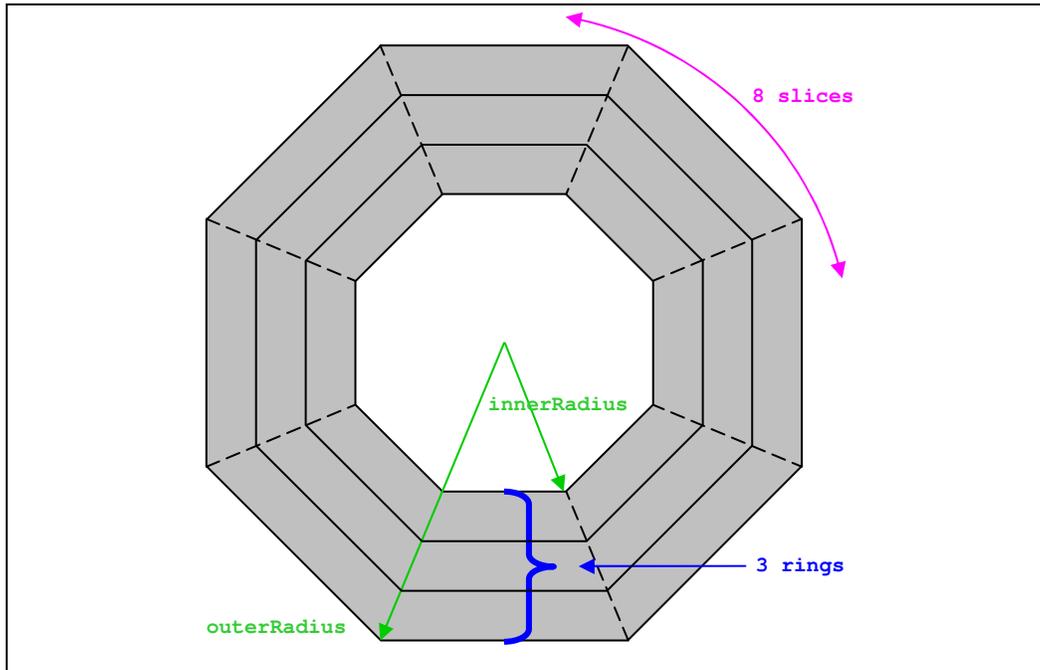
... construct this transformation matrix.

How would you draw a filled circle?

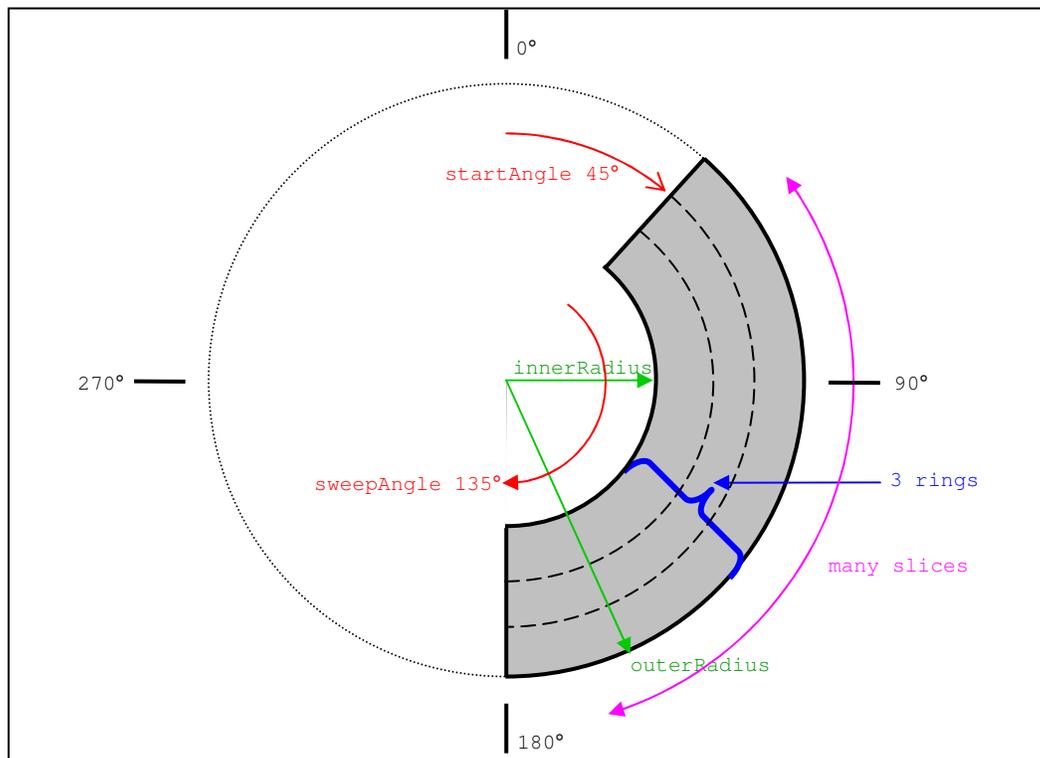


More vertices  $\Rightarrow$  closer to a smooth circle  
OpenGL: "quadric objects"

```
void gluDisk( GLUQuadricObj *obj,
             GLdouble innerRadius, GLdouble outerRadius,
             GLint slices, GLint rings );
```



```
void gluPartialDisk( GLUQuadricObj *obj,
                   GLdouble innerRadius, GLdouble outerRadius,
                   GLint slices, GLint rings,
                   GLdouble startAngle, GLdouble sweepAngle );
```



```
GLUquadricObj* gluNewQuadric( void );  
gluDeleteQuadric( GLUquadricObj *obj );  
void gluQuadricDrawStyle( GLUquadricObj *obj, GLenum style );  
                                GLU_LINE, GLU_FILL ↕
```

(Walk thru example program)

GLUT menus (walkthru example menu program #1)

Desired menu setup: a single main menu

```

Item1
Item2
Exit

```

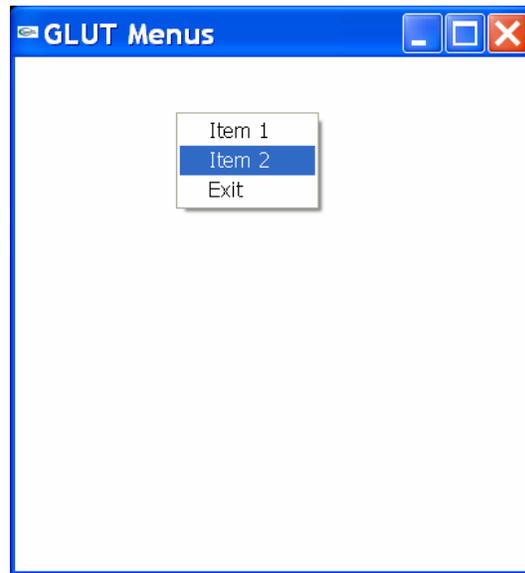
```

// menu.h

// menu item ID's,
// sequential by menu position
// starting at 1
const int ITEM1 = 1;
const int ITEM2 = 2;
const int EXIT  = 3;

// main menu handler
void main_menu( int item );

```



```

// menu.cpp

// menu setup:
void menu_init( void )
{
    /* create the main menu */
    glutCreateMenu(main_menu);
    glutAddMenuEntry("Item 1 ", ITEM1); // adds a new menu item to the
    glutAddMenuEntry("Item 2 ", ITEM2); // bottom of the current menu
    glutAddMenuEntry("Exit  ", EXIT);

    glutAttachMenu(GLUT_RIGHT_BUTTON); // menu opens on RMB press
}

// menu handler:
void main_menu( int item )
{
    switch( item )
    {
        case ITEM1: // code to handle item 1
        case ITEM2: // code to handle item 2
        case EXIT:  // code to exit
    }
}

```

## (Walkthrough example menu program #2)

Desired menu setup: a main menu with a submenu:

```

Item1
Submenu --> SubItem1
Exit      SubItem2

```

```

// menu.h
// main menu item ID's
const int ITEM1 = 1;
const int EXIT  = 2;

// submenu item ID's
const int SUBITEM1 = 1;
const int SUBITEM2 = 2;

// menu handlers
void main_menu( int item );
void sub_menu( int item );

```

```

// menu.cpp
void menu_init( void )
{
    // create the submenu FIRST
    int submenu_id = glutCreateMenu(sub_menu);
    glutAddMenuEntry("SubItem 1 ", SUBITEM1);
    glutAddMenuEntry("SubItem 2 ", SUBITEM2);

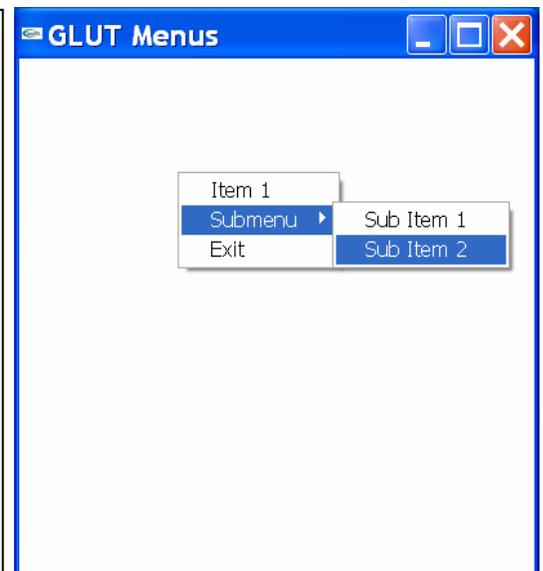
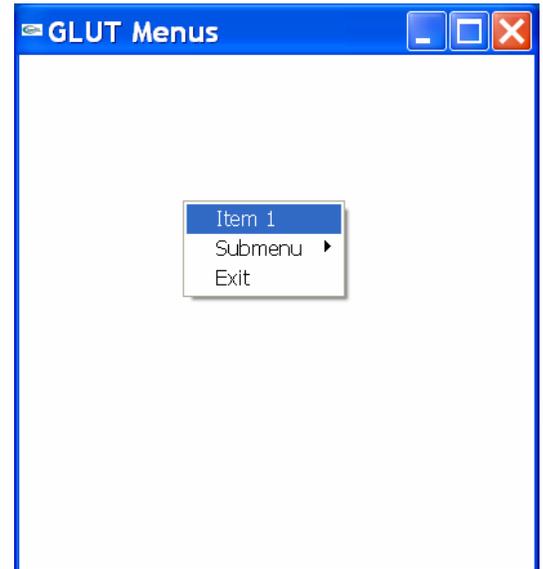
    // create the main menu
    glutCreateMenu(main_menu);
    glutAddMenuEntry("Item 1 ", ITEM1);
    glutAddSubMenu( "Submenu ", submenu_id);
    glutAddMenuEntry("Exit   ", EXIT);

    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

void main_menu( int item )
{
    switch( item )
    {
        case ITEM1: // code to handle item 1
        case EXIT:  // code to exit
    }
}

void sub_menu( int item )
{
    switch( item )
    {
        case ITEM1: // code to handle item 1
        case ITEM2: // code to handle item 2
    }
}

```



Modifying menus at run-time

```
glutDetachMenu( int button ); // no longer pop-up  
  
glutChangeToMenuEntry( int entry, char *text, int value )
```

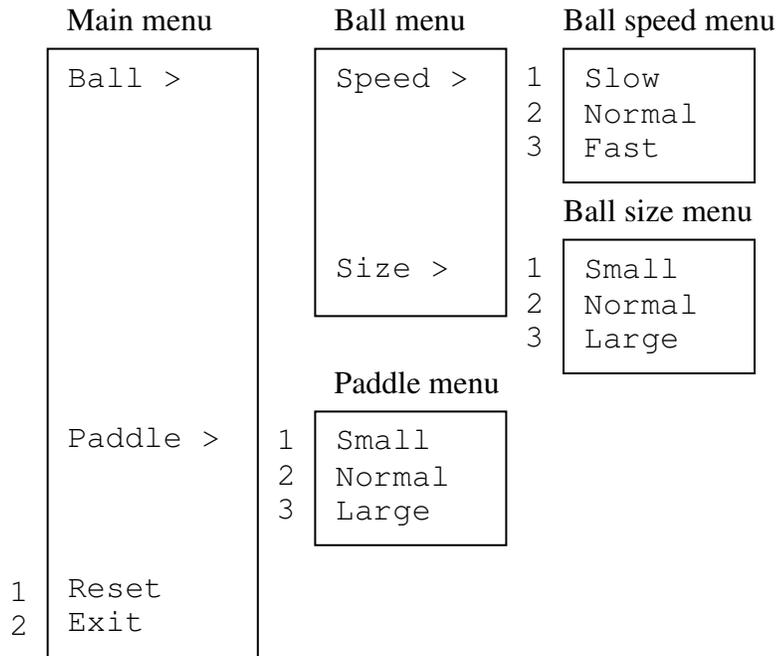
Example:

```
if( item1_selected ) // deselect it  
    glutChangeToMenuEntry(ITEM1, " Item 1", ITEM1);  
else // select it  
    glutChangeToMenuEntry(ITEM1, "* Item 1", ITEM1);  
item1_selected = !item1_selected;
```

Lab:

Make a copy of and modify your PONG I or II lab, or use the code provided.

1. Add the following menu system:



2. Implement the **Exit** and **Paddle** menu functionality.

3. Draw a ball somewhere in the viewport using `gluDisk` (it doesn't have to move!)