```
#include <stdio.h>
        // yes, I know stdio.h is not good C++, but I like the *printf()
#include <stdlib.h>
#include <ctype.h>

#define _USE_MATH_DEFINES
#include <math.h>

#ifdef WIN32
#include <windows.h>
#endif

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include "glui.h"

//
//
//      This is a sample OpenGL / GLUT / GLUI program for CS 519
//
//      The objective is to draw one of a number of 3d objects
//
//      The left mouse button allows rotation
//      The middle mouse button allows scaling
//      The glui window allows:
//              1. The 3d object to be changed
//              2. The projection to be changed
//              3. The colors to be changed
//              4. Axes to be turned on and off
//              5. The transformations to be reset
//              6. The program to quit
//
//      Author: Joe Graphics
//


//
// constants:
//
// NOTE: There are a bunch of good reasons to use const variables instead
// of #define's.  However, Visual C++ does not allow a const variable
// to be an array size or the case in a switch() statement.  So in the following,
// all constants are const variables except those which need to be array sizes
// or cases in switch() statements.  Those are #defines.  (Microsoft, please
// fix this, UNIX figured it out years ago...)
//
//


// title of these windows:

const char *WINDOWTITLE = { "OpenGL / GLUT / GLUI Sample -- Joe Graphics" };
const char *GLUITITLE   = { "User Interface Window" };


// what the glui package defines as true and false:

const int GLUITRUE  = { true  };
const int GLUIFALSE = { false };


// the escape key:

#define ESCAPE          0x1b
```

```
// initial window size:

const int INIT_WINDOW_SIZE = { 600 };



// multiplication factors for input interaction:
//  (these are known from previous experience)

const float ANGFACT = { 1. };
const float SCLFACT = { 0.005f };


// able to use the left mouse for either rotation or scaling,
// in case have only a 2-button mouse:

enum LeftButton
{
        ROTATE,
        SCALE
};


// minimum allowable scale factor:

const float MINSCALE = { 0.05f };


// active mouse buttons (or them together):

const int LEFT   = { 4 };
const int MIDDLE = { 2 };
const int RIGHT  = { 1 };


// which projection:

enum Projections
{
        ORTHO,
        PERSP
};


// which button:

enum ButtonVals
{
        RESET,
        QUIT
};


// window background color (rgba):

const float BACKCOLOR[] = { 0., 0., 0., 0. };


// color and line width for the axes:

const GLfloat AXES_COLOR[] = { 1., .5, 0. };
const GLfloat AXES_WIDTH   = { 3. };
```

```
// the objects:
// this order must match the radio button order
// these const variables are not actually used in the program
// they are just here to remind what the order must be

enum Objects
{
        SPHERE,
        BOX,
        CONE,
        TORUS,
        DODECAHEDRON,
        OCTAHEADRON,
        TETRAHEDRON,
        ICOSAHEDRON,
        TEAPOT
};


// max # of objects:

#define MAXOBJECTS              9


// the color numbers:
// this order must match the radio button order

enum Colors
{
        RED,
        YELLOW,
        GREEN,
        CYAN,
        BLUE,
        MAGENTA,
        WHITE,
        BLACK
};


// the color definitions:
// this order must match the radio button order

const GLfloat Colors[8][3] =
{
        { 1., 0., 0. },         // red
        { 1., 1., 0. },         // yellow
        { 0., 1., 0. },         // green
        { 0., 1., 1. },         // cyan
        { 0., 0., 1. },         // blue
        { 1., 0., 1. },         // magenta
        { 1., 1., 1. },         // white
        { 0., 0., 0. },         // black
};


// fog parameters:

const GLfloat FOGCOLOR[4] = { .0, .0, .0, 1. };
const GLenum  FOGMODE     = { GL_LINEAR };
const GLfloat FOGDENSITY  = { 0.30f };
const GLfloat FOGSTART    = { 1.5 };
const GLfloat FOGEND      = { 4. };
```

```
//
// non-constant global variables:
//

int     ActiveButton;           // current button that is down
GLuint  AxesList;               // list to hold the axes
int     AxesOn;                 // ON or OFF
int     Debug;                  // ON means print debug info
int     DepthCueOn;             // TRUE means to use intensity depth cueing
GLUI *  Glui;                   // instance of glui window
int     GluiWindow;             // the glut id for the glui window
int     LeftButton;             // either ROTATE or SCALE
GLuint  Lists[MAXOBJECTS];      // object display lists
int     MainWindow;             // window id for main graphics window
GLfloat RotMatrix[4][4];        // set by glui rotation widget
float   Scale, Scale2;          // scaling factors
int     WhichColor;             // index into Colors[]
int     WhichObject;            // object index to display
int     WhichProjection;        // ORTHO or PERSP
int     Xmouse, Ymouse;         // mouse values
float   Xrot, Yrot;             // rotation angles in degrees
float   TransXYZ[3];            // set by glui translation widgets


//
// function prototypes:
//

void    Animate( void );
void    Axes( float );
void    Buttons( int );
void    Display( void );
void    DoRasterString( float, float, float, char * );
void    DoStrokeString( float, float, float, float, char * );
float   ElapsedSeconds( void );
void    HsvRgb( float[3], float [3] );
void    InitGlui( void );
void    InitGraphics( void );
void    InitLists( void );
void    Keyboard( unsigned char, int, int );
void    MouseButton( int, int, int, int );
void    MouseMotion( int, int );
void    Reset( void );
void    Resize( int, int );
void    Visibility( int );


//
// main program:
//

int
main( int argc, char *argv[] )
{
        // turn on the glut package:
        // (do this before checking argc and argv since it might
        // pull some command line arguments out)

        glutInit( &argc, argv );


        // setup all the graphics stuff:
```

```
	InitGraphics();


	// create the display structures that will not change:

	InitLists();


	// init all the global variables used by Display():
	// this will also post a redisplay
	// it is important to call this before InitGlui():
	// so that the variables that glui will control are correct
	// when each glui widget is created

	Reset();


	// setup all the user interface stuff:

	InitGlui();


	// draw the scene once and wait for some interaction:
	// (will never return)

	glutMainLoop();


	// this is here to make the compiler happy:

	return 0;
}



//
// this is where one would put code that is to be called
// everytime the glut main loop has nothing to do
//
// this is typically where animation parameters are set
//
// do not call Display() from here -- let glutMainLoop() do it
//

void
Animate( void )
{
	// put animation stuff in here -- change some global variables
	// for Display() to find:



	// force a call to Display() next time it is convenient:

	glutSetWindow( MainWindow );
	glutPostRedisplay();
}



//
// glui buttons callback:
//

void
```

```
Buttons( int id )
{
	switch( id )
	{
		case RESET:
			Reset();
			Glui->sync_live();
			glutSetWindow( MainWindow );
			glutPostRedisplay();
			break;

		case QUIT:
			// gracefully close the glui window:
			// gracefully close out the graphics:
			// gracefully close the graphics window:
			// gracefully exit the program:

			Glui->close();
			glFinish();
			glutDestroyWindow( MainWindow );
			exit( 0 );

		default:
			fprintf( stderr, "Don't know what to do with Button ID %d\n", id );
	}

}



//
// draw the complete scene:
//

void
Display( void )
{
	GLsizei vx, vy, v;		// viewport dimensions
	GLint xl, yb;		// lower-left corner of viewport
	GLfloat scale2;		// real glui scale factor

	if( Debug )
	{
		fprintf( stderr, "Display\n" );
	}


	// set which window we want to do the graphics into:

	glutSetWindow( MainWindow );


	// erase the background:

	glDrawBuffer( GL_BACK );
	glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
	glEnable( GL_DEPTH_TEST );


	// specify shading to be flat:

	glShadeModel( GL_FLAT );


	// set the viewport to a square centered in the window:
```

```
	vx = glutGet( GLUT_WINDOW_WIDTH );
	vy = glutGet( GLUT_WINDOW_HEIGHT );
	v = vx < vy ? vx : vy;			// minimum dimension
	xl = ( vx - v ) / 2;
	yb = ( vy - v ) / 2;
	glViewport( xl, yb,  v, v );


	// set the viewing volume:
	// remember that the Z clipping  values are actually
	// given as DISTANCES IN FRONT OF THE EYE
	// ONLY USE gluOrtho2D() IF YOU ARE DOING 2D !

	glMatrixMode( GL_PROJECTION );
	glLoadIdentity();

	if( WhichProjection == ORTHO )
		glOrtho( -3., 3.,     -3., 3.,     0.1, 1000. );
	else
		gluPerspective( 90., 1.,	0.1, 1000. );


	// place the objects into the scene:

	glMatrixMode( GL_MODELVIEW );
	glLoadIdentity();


	// set the eye position, look-at position, and up-vector:
	// IF DOING 2D, REMOVE THIS -- OTHERWISE ALL YOUR 2D WILL DISAPPEAR !

	gluLookAt( 0., 0., 3.,     0., 0., 0.,     0., 1., 0. );


	// translate the objects in the scene:
	// note the minus sign on the z value
	// this is to make the appearance of the glui z translate
	// widget more intuitively match the translate behavior
	// DO NOT TRANSLATE IN Z IF YOU ARE DOING 2D !

	glTranslatef( (GLfloat)TransXYZ[0], (GLfloat)TransXYZ[1], -(GLfloat)TransXYZ[2] );


	// rotate the scene:
	// DO NOT ROTATE (EXCEPT ABOUT Z) IF YOU ARE DOING 2D !

	glRotatef( (GLfloat)Yrot, 0., 1., 0. );
	glRotatef( (GLfloat)Xrot, 1., 0., 0. );
	glMultMatrixf( (const GLfloat *) RotMatrix );


	// scale the scene:

	glScalef( (GLfloat)Scale, (GLfloat)Scale, (GLfloat)Scale );
	scale2 = 1. + Scale2;		// because glui translation starts at 0.
	if( scale2 < MINSCALE )
		scale2 = MINSCALE;
	glScalef( (GLfloat)scale2, (GLfloat)scale2, (GLfloat)scale2 );


	// set the fog parameters:
	// DON'T NEED THIS IF DOING 2D !

	if( DepthCueOn )
```

```
	{
		glFogi( GL_FOG_MODE, FOGMODE );
		glFogfv( GL_FOG_COLOR, FOGCOLOR );
		glFogf( GL_FOG_DENSITY, FOGDENSITY );
		glFogf( GL_FOG_START, FOGSTART );
		glFogf( GL_FOG_END, FOGEND );
		glEnable( GL_FOG );
	}
	else
	{
		glDisable( GL_FOG );
	}


	// possibly draw the axes:

	if( AxesOn )
		glCallList( AxesList );


	// set the color of the object:

	glColor3fv( Colors[WhichColor] );


	// draw the object:

	glCallList( Lists[WhichObject] );


	// draw some gratuitous text that just rotates on top of the scene:

	glDisable( GL_DEPTH_TEST );
	glColor3f( 0., 1., 1. );
	DoRasterString( 0., 1., 0., "Text That Moves" );


	// draw some gratuitous text that is fixed on the screen:
	// the projection matrix is reset to define a scene whose
	// world coordinate system goes from 0-100 in each axis
	// this is called "percent units", and is just a convenience
	// the modelview matrix is reset to identity as we don't
	// want to transform these coordinates

	glDisable( GL_DEPTH_TEST );
	glMatrixMode( GL_PROJECTION );
	glLoadIdentity();
	gluOrtho2D( 0., 100.,     0., 100. );
	glMatrixMode( GL_MODELVIEW );
	glLoadIdentity();
	glColor3f( 1., 1., 1. );
	DoRasterString( 5., 5., 0., "Text That Doesn't" );


	// swap the double-buffered framebuffers:

	glutSwapBuffers();


	// be sure the graphics buffer has been sent:

	glFlush();
}
```

```
//
// use glut to display a string of characters using a raster font:
//

void
DoRasterString( float x, float y, float z, char *s )
{
	char c;			// one character to print

	glRasterPos3f( (GLfloat)x, (GLfloat)y, (GLfloat)z );
	for( ; ( c = *s ) != '\0'; s++ )
	{
		glutBitmapCharacter( GLUT_BITMAP_TIMES_ROMAN_24, c );
	}
}



//
// use glut to display a string of characters using a stroke font:
//

void
DoStrokeString( float x, float y, float z, float ht, char *s )
{
	char c;			// one character to print
	float sf;		// the scale factor

	glPushMatrix();
		glTranslatef( (GLfloat)x, (GLfloat)y, (GLfloat)z );
		sf = ht / ( 119.05 + 33.33 );
		glScalef( (GLfloat)sf, (GLfloat)sf, (GLfloat)sf );
		for( ; ( c = *s ) != '\0'; s++ )
		{
			glutStrokeCharacter( GLUT_STROKE_ROMAN, c );
		}
	glPopMatrix();
}



//
// return the number of seconds since the start of the program:
//

float
ElapsedSeconds( void )
{
	int ms;


	// get # of milliseconds since the start of the program:

	ms = glutGet( GLUT_ELAPSED_TIME );


	// convert it to seconds:

	return (float)ms / 1000.;
}



//
```

```
// initialize the glui window:
//

void
InitGlui( void )
{
	GLUI_Panel *panel, *bigpanel;
	GLUI_RadioGroup *group;
	GLUI_Rotation *rot;
	GLUI_Translation *trans, *scale;

	if( Debug )
		fprintf( stderr, "InitGlui\n" );


	// setup the glui window:

	glutInitWindowPosition( INIT_WINDOW_SIZE + 50, 0 );
	Glui = GLUI_Master.create_glui( (char *) GLUITITLE );


	Glui->add_statictext( (char *) GLUITITLE );
	Glui->add_separator();

	Glui->add_checkbox( "Axes", &AxesOn );

	Glui->add_checkbox( "Perspective", &WhichProjection );

	Glui->add_checkbox( "Intensity Depth Cue", &DepthCueOn );

	bigpanel = Glui->add_panel( "" );

	panel = Glui->add_panel_to_panel( bigpanel, "Object Selection" );
		group = Glui->add_radiogroup_to_panel( panel, &WhichObject );
			Glui->add_radiobutton_to_group( group, "Sphere" );
			Glui->add_radiobutton_to_group( group, "Cube" );
			Glui->add_radiobutton_to_group( group, "Cone" );
			Glui->add_radiobutton_to_group( group, "Torus" );
			Glui->add_radiobutton_to_group( group, "Dodecahedron" );
			Glui->add_radiobutton_to_group( group, "Octahedron" );
			Glui->add_radiobutton_to_group( group, "Tetrahedron" );
			Glui->add_radiobutton_to_group( group, "Icosahedron" );
			Glui->add_radiobutton_to_group( group, "Teapot" );

	Glui->add_column_to_panel( bigpanel, GLUITRUE );

	panel = Glui->add_panel_to_panel( bigpanel, "Object Color" );
		group = Glui->add_radiogroup_to_panel( panel, &WhichColor );
			Glui->add_radiobutton_to_group( group, "Red" );
			Glui->add_radiobutton_to_group( group, "Yellow" );
			Glui->add_radiobutton_to_group( group, "Green" );
			Glui->add_radiobutton_to_group( group, "Cyan" );
			Glui->add_radiobutton_to_group( group, "Blue" );
			Glui->add_radiobutton_to_group( group, "Magenta" );
			Glui->add_radiobutton_to_group( group, "White" );
			Glui->add_radiobutton_to_group( group, "Black" );

	panel = Glui->add_panel( "Object Transformation" );

		rot = Glui->add_rotation_to_panel( panel, "Rotation", (float *) RotMatrix );

		// allow the object to be spun via the glui rotation widget:

		rot->set_spin( 1.0 );
```

```
                Glui->add_column_to_panel( panel, GLUIFALSE );
                scale = Glui->add_translation_to_panel( panel, "Scale",  GLUI_TRANSLATION_Y ,
 &Scale2 );
                scale->set_speed( 0.01f );

                Glui->add_column_to_panel( panel, FALSE );
                trans = Glui->add_translation_to_panel( panel, "Trans XY", GLUI_TRANSLATION_X
Y, &TransXYZ[0] );
                trans->set_speed( 0.1f );

                Glui->add_column_to_panel( panel, FALSE );
                trans = Glui->add_translation_to_panel( panel, "Trans Z",  GLUI_TRANSLATION_Z
 , &TransXYZ[2] );
                trans->set_speed( 0.1f );

        Glui->add_checkbox( "Debug", &Debug );


        panel = Glui->add_panel( "", FALSE );

        Glui->add_button_to_panel( panel, "Reset", RESET, (GLUI_Update_CB) Buttons );

        Glui->add_column_to_panel( panel, FALSE );

        Glui->add_button_to_panel( panel, "Quit", QUIT, (GLUI_Update_CB) Buttons );


        // tell glui what graphics window it needs to post a redisplay to:

        Glui->set_main_gfx_window( MainWindow );


        // set the graphics window's idle function:

        GLUI_Master.set_glutIdleFunc( NULL );
}



//
// initialize the glut and OpenGL libraries:
//      also setup display lists and callback functions
//

void
InitGraphics( void )
{
        if( Debug )
                fprintf( stderr, "InitGraphics\n" );


        // setup the display mode:
        // ( *must* be done before call to glutCreateWindow() )
        // ask for color, double-buffering, and z-buffering:

        glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );


        // set the initial window configuration:

        glutInitWindowPosition( 0, 0 );
        glutInitWindowSize( INIT_WINDOW_SIZE, INIT_WINDOW_SIZE );
```

```
	// open the window and set its title:

	MainWindow = glutCreateWindow( WINDOWTITLE );
	glutSetWindowTitle( WINDOWTITLE );


	// setup the clear values:

	glClearColor( BACKCOLOR[0], BACKCOLOR[1], BACKCOLOR[2], BACKCOLOR[3] );


	// setup the callback routines:


	// DisplayFunc -- redraw the window
	// ReshapeFunc -- handle the user resizing the window
	// KeyboardFunc -- handle a keyboard input
	// MouseFunc -- handle the mouse button going down or up
	// MotionFunc -- handle the mouse moving with a button down
	// PassiveMotionFunc -- handle the mouse moving with a button up
	// VisibilityFunc -- handle a change in window visibility
	// EntryFunc    -- handle the cursor entering or leaving the window
	// SpecialFunc -- handle special keys on the keyboard
	// SpaceballMotionFunc -- handle spaceball translation
	// SpaceballRotateFunc -- handle spaceball rotation
	// SpaceballButtonFunc -- handle spaceball button hits
	// ButtonBoxFunc -- handle button box hits
	// DialsFunc -- handle dial rotations
	// TabletMotionFunc -- handle digitizing tablet motion
	// TabletButtonFunc -- handle digitizing tablet button hits
	// MenuStateFunc -- declare when a pop-up menu is in use
	// TimerFunc -- trigger something to happen a certain time from now
	// IdleFunc -- what to do when nothing else is going on

	glutSetWindow( MainWindow );
	glutDisplayFunc( Display );
	glutReshapeFunc( Resize );
	glutKeyboardFunc( Keyboard );
	glutMouseFunc( MouseButton );
	glutMotionFunc( MouseMotion );
	glutPassiveMotionFunc( NULL );
	glutVisibilityFunc( Visibility );
	glutEntryFunc( NULL );
	glutSpecialFunc( NULL );
	glutSpaceballMotionFunc( NULL );
	glutSpaceballRotateFunc( NULL );
	glutSpaceballButtonFunc( NULL );
	glutButtonBoxFunc( NULL );
	glutDialsFunc( NULL );
	glutTabletMotionFunc( NULL );
	glutTabletButtonFunc( NULL );
	glutMenuStateFunc( NULL );
	glutTimerFunc( 0, NULL, 0 );

	// DO NOT SET THE GLUT IDLE FUNCTION HERE !!
	// glutIdleFunc( NULL );
	// let glui take care of it in InitGlui()
}



//
// initialize the display lists that will not change:
//
```

```
void
InitLists( void )
{
	if( Debug )
		fprintf( stderr, "InitLists\n" );

	// create the objects:

	Lists[0] = glGenLists( 1 );
	glNewList( Lists[0], GL_COMPILE );
		glutWireSphere( 1.0, 40, 40 );
	glEndList();

	Lists[1] = glGenLists( 1 );
	glNewList( Lists[1], GL_COMPILE );
		glutWireCube( 1.5 );
	glEndList();

	Lists[2] = glGenLists( 1 );
	glNewList( Lists[2], GL_COMPILE );
		glutWireCone( 1.0, 1.5, 40, 40 );
	glEndList();

	Lists[3] = glGenLists( 1 );
	glNewList( Lists[3], GL_COMPILE );
		glutWireTorus( 0.5, 0.75, 40, 40 );
	glEndList();

	Lists[4] = glGenLists( 1 );
	glNewList( Lists[4], GL_COMPILE );
		glPushMatrix();
			glScalef( 0.75, 0.75, 0.75 );
			glutWireDodecahedron();
		glPopMatrix();
	glEndList();

	Lists[5] = glGenLists( 1 );
	glNewList( Lists[5], GL_COMPILE );
		glutWireOctahedron();
	glEndList();

	Lists[6] = glGenLists( 1 );
	glNewList( Lists[6], GL_COMPILE );
		glutWireTetrahedron();
	glEndList();

	Lists[7] = glGenLists( 1 );
	glNewList( Lists[7], GL_COMPILE );
		glutWireIcosahedron();
	glEndList();

	Lists[8] = glGenLists( 1 );
	glNewList( Lists[8], GL_COMPILE );
		glutWireTeapot( 1.0 );
	glEndList();


	// create the axes:

	AxesList = glGenLists( 1 );
	glNewList( AxesList, GL_COMPILE );
		glColor3fv( AXES_COLOR );
		glLineWidth( AXES_WIDTH );
			Axes( 1.5 );
```

```
                glLineWidth( 1. );
        glEndList();
}



//
// the keyboard callback:
//

void
Keyboard( unsigned char c, int x, int y )
{
        if( Debug )
                fprintf( stderr, "Keyboard: '%c' (0x%0x)\n", c, c );

        switch( c )
        {
                case 'o':
                case 'O':
                        WhichProjection = ORTHO;
                        break;

                case 'p':
                case 'P':
                        WhichProjection = PERSP;
                        break;

                case 'q':
                case 'Q':
                case ESCAPE:
                        Buttons( QUIT );        // will not return here
                        break;                  // happy compiler

                case 'r':
                case 'R':
                        LeftButton = ROTATE;
                        break;

                case 's':
                case 'S':
                        LeftButton = SCALE;
                        break;

                default:
                        fprintf( stderr, "Don't know what to do with keyboard hit: '%c' (0x%0
x)\n", c, c );
        }


        // synchronize the GLUI display with the variables:

        Glui->sync_live();


        // force a call to Display():

        glutSetWindow( MainWindow );
        glutPostRedisplay();
}



//
// called when the mouse button transitions down or up:
```

```
//

void
MouseButton( int button, int state, int x, int y )
{
	int b;			// LEFT, MIDDLE, or RIGHT

	if( Debug )
		fprintf( stderr, "MouseButton: %d, %d, %d, %d\n", button, state, x, y );


	// get the proper button bit mask:

	switch( button )
	{
		case GLUT_LEFT_BUTTON:
			b = LEFT;		break;

		case GLUT_MIDDLE_BUTTON:
			b = MIDDLE;		break;

		case GLUT_RIGHT_BUTTON:
			b = RIGHT;		break;

		default:
			b = 0;
			fprintf( stderr, "Unknown mouse button: %d\n", button );
	}


	// button down sets the bit, up clears the bit:

	if( state == GLUT_DOWN )
	{
		Xmouse = x;
		Ymouse = y;
		ActiveButton |= b;		// set the proper bit
	}
	else
	{
		ActiveButton &= ~b;		// clear the proper bit
	}
}



//
// called when the mouse moves while a button is down:
//

void
MouseMotion( int x, int y )
{
	int dx, dy;		// change in mouse coordinates

	if( Debug )
		fprintf( stderr, "MouseMotion: %d, %d\n", x, y );


	dx = x - Xmouse;		// change in mouse coords
	dy = y - Ymouse;

	if( ActiveButton & LEFT )
	{
		switch( LeftButton )
```

```
                {
                        case ROTATE:
                                Xrot += ( ANGFACT*dy );
                                Yrot += ( ANGFACT*dx );
                                break;

                        case SCALE:
                                Scale += SCLFACT * (float) ( dx - dy );
                                if( Scale < MINSCALE )
                                        Scale = MINSCALE;
                                break;
                }
        }


        if( ActiveButton & MIDDLE )
        {
                Scale += SCLFACT * (float) ( dx - dy );

                // keep object from turning inside-out or disappearing:

                if( Scale < MINSCALE )
                        Scale = MINSCALE;
        }

        Xmouse = x;                     // new current position
        Ymouse = y;

        glutSetWindow( MainWindow );
        glutPostRedisplay();
}



//
// reset the transformations and the colors:
//
// this only sets the global variables --
// the glut main loop is responsible for redrawing the scene
//

void
Reset( void )
{
        ActiveButton = 0;
        AxesOn = GLUITRUE;
        Debug = GLUIFALSE;
        DepthCueOn = FALSE;
        LeftButton = ROTATE;
        Scale  = 1.0;
        Scale2 = 0.0;           // because add 1. to it in Display()
        WhichColor = WHITE;
        WhichObject = SPHERE;
        WhichProjection = PERSP;
        Xrot = Yrot = 0.;
        TransXYZ[0] = TransXYZ[1] = TransXYZ[2] = 0.;

                        RotMatrix[0][1] = RotMatrix[0][2] = RotMatrix[0][3] = 0.;
        RotMatrix[1][0]                 = RotMatrix[1][2] = RotMatrix[1][3] = 0.;
        RotMatrix[2][0] = RotMatrix[2][1]                 = RotMatrix[2][3] = 0.;
        RotMatrix[3][0] = RotMatrix[3][1] = RotMatrix[3][3]                 = 0.;
        RotMatrix[0][0] = RotMatrix[1][1] = RotMatrix[2][2] = RotMatrix[3][3] = 1.;
}
```

```
//
// called when user resizes the window:
//

void
Resize( int width, int height )
{
        if( Debug )
                fprintf( stderr, "ReSize: %d, %d\n", width, height );

        // don't really need to do anything since window size is
        // checked each time in Display():

        glutSetWindow( MainWindow );
        glutPostRedisplay();
}


//
// handle a change to the window's visibility:
//

void
Visibility ( int state )
{
        if( Debug )
                fprintf( stderr, "Visibility: %d\n", state );

        if( state == GLUT_VISIBLE )
        {
                glutSetWindow( MainWindow );
                glutPostRedisplay();
        }
        else
        {
                // could optimize by keeping track of the fact
                // that the window is not visible and avoid
                // animating or redrawing it ...
        }
}







// the stroke characters 'X' 'Y' 'Z' :

static float xx[] = {
                0., 1., 0., 1.
              };

static float xy[] = {
                -.5, .5, .5, -.5
              };

static int xorder[] = {
                1, 2, -3, 4
                };


static float yx[] = {
                0., 0., -.5, .5
```

```
		};

static float yy[] = {
		0., .6, 1., 1.
		};

static int yorder[] = {
		1, 2, 3, -2, 4
		};


static float zx[] = {
		1., 0., 1., 0., .25, .75
		};

static float zy[] = {
		.5, .5, -.5, -.5, 0., 0.
		};

static int zorder[] = {
		1, 2, 3, 4, -5, 6
		};


// fraction of the length to use as height of the characters:

#define LENFRAC		0.10


// fraction of length to use as start location of the characters:

#define BASEFRAC	1.10


//
//	Draw a set of 3D axes:
//	(length is the axis length in world coordinates)
//

void
Axes( float length )
{
	int i, j;			// counters
	float fact;			// character scale factor
	float base;			// character start location


	glBegin( GL_LINE_STRIP );
		glVertex3f( length, 0., 0. );
		glVertex3f( 0., 0., 0. );
		glVertex3f( 0., length, 0. );
	glEnd();
	glBegin( GL_LINE_STRIP );
		glVertex3f( 0., 0., 0. );
		glVertex3f( 0., 0., length );
	glEnd();

	fact = LENFRAC * length;
	base = BASEFRAC * length;

	glBegin( GL_LINE_STRIP );
		for( i = 0; i < 4; i++ )
		{
			j = xorder[i];
			if( j < 0 )
```

```
                        {

                                glEnd();
                                glBegin( GL_LINE_STRIP );
                                j = -j;
                        }
                        j--;
                        glVertex3f( base + fact*xx[j], fact*xy[j], 0.0 );
                }
        glEnd();

        glBegin( GL_LINE_STRIP );
                for( i = 0; i < 5; i++ )
                {
                        j = yorder[i];
                        if( j < 0 )
                        {

                                glEnd();
                                glBegin( GL_LINE_STRIP );
                                j = -j;
                        }
                        j--;
                        glVertex3f( fact*yx[j], base + fact*yy[j], 0.0 );
                }
        glEnd();

        glBegin( GL_LINE_STRIP );
                for( i = 0; i < 6; i++ )
                {
                        j = zorder[i];
                        if( j < 0 )
                        {

                                glEnd();
                                glBegin( GL_LINE_STRIP );
                                j = -j;
                        }
                        j--;
                        glVertex3f( 0.0, fact*zy[j], base + fact*zx[j] );
                }
        glEnd();

}

//
// routine to convert HSV to RGB
//
// Reference:  Foley, van Dam, Feiner, Hughes,
//              "Computer Graphics Principles and Practices,"
//              Additon-Wesley, 1990, pp592-593.
//
//


void
HsvRgb( float hsv[3], float rgb[3] )
{
        float h, s, v;                  // hue, sat, value
        float r, g, b;                  // red, green, blue
        float i, f, p, q, t;            // interim values


        // guarantee valid input:
```

```
	h = hsv[0] / 60.;
	while( h >= 6. )	h -= 6.;
	while( h <  0. )	h += 6.;

	s = hsv[1];
	if( s < 0. )
		s = 0.;
	if( s > 1. )
		s = 1.;

	v = hsv[2];
	if( v < 0. )
		v = 0.;
	if( v > 1. )
		v = 1.;


	// if sat==0, then is a gray:

	if( s == 0.0 )
	{
		rgb[0] = rgb[1] = rgb[2] = v;
		return;
	}


	// get an rgb from the hue itself:

	i = floor( h );
	f = h - i;
	p = v * ( 1. - s );
	q = v * ( 1. - s*f );
	t = v * ( 1. - ( s * (1.-f) ) );

	switch( (int) i )
	{
		case 0:
			r = v;	g = t;	b = p;
			break;

		case 1:
			r = q;	g = v;	b = p;
			break;

		case 2:
			r = p;	g = v;	b = t;
			break;

		case 3:
			r = p;	g = q;	b = v;
			break;

		case 4:
			r = t;	g = p;	b = v;
			break;

		case 5:
			r = v;	g = p;	b = q;
			break;
	}


	rgb[0] = r;
	rgb[1] = g;
	rgb[2] = b;
```

```
}
```