

## Documentation for opengl2.h and opengl2.c

Steven Andrews, © 2005

See the document “LibDoc” for general information about this and other libraries.

### Header file

```

#ifndef __opengl2_h
#define __opengl2_h

/* If the OpenGL graphics libraries are unavailable, comment out the following
lines (most functions here will become non-active). Also, note that different
names may be required for OpenGL header files, such as <GL/gl.h> */
#include <gl.h>
#include <glut.h>

/* If the libtiff library is unavailable, comment out the following line. */
#include "tiffio.h"

void gl2Initialize(float xlo,float xhi,float ylo,float yhi,float zlo,float
zhi);
void gl2glutInit(int *argc,char **argv);
int gl2State(int state);
float gl2GetNumber(char *variable);
int gl2SetOptionInt(char *option,int value);
float gl2SetOptionFlt(char *option,float value);
char *gl2SetOptionStr(char *option,char *value);
void gl2SetKeyPush(unsigned char key);
void gl2SetColor(char c);

float gl2FindRotate(float *vect1,float *vect2,float *axis);
double gl2FindRotateD(double *vect1,double *vect2,double *axis);
void gl2DrawBox(float *pt1,float *pt2,int dim);
void gl2DrawBoxD(double *pt1,double *pt2,int dim);
void gl2DrawGrid(float *pt1,float *pt2,int *n,int dim);
void gl2DrawGridD(double *pt1,double *pt2,int *n,int dim);
void gl2DrawCircle(float *cent,float radius,int slices,char style);
void gl2DrawCircleD(double *cent,double radius,int slices,char style);
void gl2DrawArc(float *cent,float radius,float theta1,float theta2,int
slices,char style);
void gl2DrawArcD(double *cent,double radius,double theta1,double theta2,int
slices,char style);
void gl2DrawHemisphere(float radius,int slices,int stacks,int frontin);
void gl2DrawCylinder(float baseRadius,float topRadius,float height,int
slices,int stacks,int frontin);
void gl2DrawSphere(float radius,int slices,int stacks,int frontin);
void gl2DrawEcoli(float radius,float length,int slices,int stacks,int frontin);
void gl2PlotData(float *xdata,float *ydata,int nx,int nycol,char *style);
void gl2PlotPts(float **data,int *ser,int nser,int npts,float **color,float
*size,char style);
void gl2PlotPtsD(double **data,int *ser,int nser,int npts,double **color,double
*size,char style);

```

```
void gl2PlotSurf(float *xdata, float *ydata, float **zdata, int nx, int ny, int
nz, char *style);
```

```
#endif
```

Requires: <math.h>, <stdio.h>, <gl.h>, <glut.h>, <stdlib.h>, <string.h>,  
"opengl2.h", "math2.h", "random.h"

Example program: smoldyn.c

History: Started 12/02, modified substantially and documented 7/03. Some testing.  
Modified KeyPush 9/17/03. Added ability to save TIFF files 3/19/04. Added  
panning 3/23/04. Changed 2-D graphics 6/11/04. Made minor changes and  
documented 10/20/04. Added gl2SetKeyPush and gl2PlotPts, modified  
SpecialKeyPush, and made more minor changes 1/05. 3-D graphics now need to  
use glClearColor(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT). Added gl2GetNumber  
2/10/06. Added gl2DrawCircle 2/20/06. Modified gl2DrawCircle and added  
gl2DrawArc 12/2/06; also added frontin parameter to several functions. Changed  
gl2DrawBox 12/7/06 and added gl2DrawGrid. Added 2-D panning and zooming  
2/22/07. Made robust to missing TIFF or OpenGL libraries 9/16/07.

## Introduction

These routines provide easy use of the OpenGL graphics libraries for the situation in which an object is to be viewed and reoriented by the user. This library was written specifically for a few uses by the *Smoldyn* program and is thus likely to be changed significantly in the near future as I encounter more graphical needs.

## Files, linking, and compiling

Proper linking and compiling seem to be among the more mystical aspects of programming. My computer has 7 files all called "gl.h," of which at least one works properly and at least one does not work at all. I use the OpenGL files that I downloaded from the Apple website (which is in my Applications folder). To get the compiler to locate these files, an access path is declared for system files that points to this folder; it is listed before a path for the compiler folder because there are nonfunctional OpenGL files in there. Because this is a system path, OpenGL headers are #included with brackets rather than quotes. The following files are needed in my CodeWarrior projects: OpenGLLibraryStub, OpenGLMemoryStub, carbon\_glut.lib, OpenGLUtilityStub, Carbon.r. On my computer, the first four are located in Applications:openGL:Headers, and Applications:openGL:Libraries. Portions of the program that calls OpenGL routines directly need the lines: #include <gl.h> and #include <glut.h>.

This library is able to save drawings as TIFF files, which requires that a lot of files get added to a project. The sole access point is the header file tiffio.h, which is in Applications:tiff-v3.6.1:libtiff. This is declared as a user access path because it is not a standard system library. Quite a lot of C code is required as well, which I have combined together into a folder called tifflibrary, in the same folder as tiffio.h. All the TIFF code files were written by Sam Leffler and can be downloaded from <http://www.libtiff.org>. The TIFF code seems to be vastly larger than should be needed for just saving TIFF files, but it is so interconnected that I was unable to prune it down to a reasonable size. If the libtiff library is unavailable, just comment out the line "#include "tiff.io"" in the opengl2.h file and everything should work well, except that images cannot be saved.

See the Smoldyn documentation for additional information on compiling with the libtiff and OpenGL libraries, especially for other platforms.

## Using opengl2

Part of the power and portability of OpenGL is in the glut library, which takes care of a lot of window and event handling in a hardware-independent fashion. However, to work properly, it requires a significant re-arrangement of the standard structure of a C program: rather than high level functions always calling lower level functions, the OpenGL framework takes care of program control and calls previously registered call-back functions when certain events occur. Because of this, the standard flow of information from one function to another is interrupted and can only be solved by using global variables. Many global variables are used within this library for graphics parameters, which are listed below; their scope is global within the library file, but they are not available elsewhere. Also, it is suggested that the entire state of the main program is encapsulated in a single data structure, which is made global to the main program and to drawing routines.

Use this library by first calling `gl2Initialize` with the outside edges of the object to be shown. Then, call `glutDisplayFunc` with the call-back function that is supposed to render the scene (often called `RenderScene`). Then, call `glutTimerFunc` with the call-back function that executes the events that are animated (often called `TimerFunction`). Then, turn control over to OpenGL by calling `glutMainLoop`. From here on, OpenGL calls back to the scene rendering function for redraws and to the timer function for animation and other program execution. OpenGL also calls back to some internal routines in this library for window size changes and key presses. When the program terminates, control is returned from the OpenGL framework, allowing memory to be freed. Here is a typical program layout:

```
#include <gl.h>
#include <glut.h>
#include "opengl2.h"
#define REDRAW 100          // iterations between redraws
#define TEXTOUT 10000     // iterations between text output

typedef struct mystatestruct {
    float x[100];
    float y[100];
    float time;
    float dt; } *stateptr;

stateptr State;

void RenderScene();
void TimerFunction(int er);
stateptr loadstate();
void freestate(stateptr state);
int runtimestep(stateptr state);

stateptr loadstate {
    stateptr state;

    state=(stateptr)malloc(sizeof(mystatestruct));
    if(!state) return NULL;
    initialize state
    return state; }
```

```

void freestate(stateptr state) {
    free(state);
    return; }

int runtimestep(stateptr state) {
    int er;

    er=0;
    execute one time step, set er to 1 if simulation should terminate
    state->time+=state->dt;
    return er; }

void RenderScene() {
    stateptr state;

    state=State;
    if(dimension<3) glClear(GL_COLOR_BUFFER_BIT);
    else glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    gl2PlotData(state->x,state->y,100,1,"-1g");
    glutSwapBuffers();
    return; }

void TimerFunction(int er) {
    stateptr state;
    static int it=0;

    if(!er&&gl2State(-1)==0) {
        state=State;
        if(!(it%REDRAW)) glutPostRedisplay();
        if(!(it%TEXTOUT)) printf("time = %f\n",state->time);
        er=runtimestep(state);
        it++; }
    else glutPostRedisplay();
    glutTimerFunc(0,TimerFunction,er);
    return; }

int main(void) {
    stateptr state;

    state=loadstate();
    if(!state) return 0;
    State=state;
    gl2Initialize(-10,10,-10,10,0,0);
    glClearColor(1,1,1,1);
    glutDisplayFunc(RenderScene);
    glutTimerFunc(0,TimerFunction,0);
    glutMainLoop();
    freestate(state);
    printf("Press command-q to exit.\n");
}

```

```
return 0; }
```

## Manipulating the picture

Several key events are monitored which can be used to manipulate the picture. The space bar toggles the mode of the `gl2state` between 0 and 1, 'Q' puts it irreversibly into state 2, and 'T' writes the current screen output to a TIFF file. The intent is that the main program is in pause mode when that state is 1, continues when it equals 0, and quits in a normal fashion when it equals 2, although the execution of this functionality is up to the main program.

<u>Key press</u>	<u>dimensions</u>	<u>function</u>
space	1,2,3	toggle pause mode between on and off
Q	1,2,3	quit
T	1,2,3	save image as TIFF file
0	1,2,3	reset view to default
arrows	3	rotate object
shift + arrows	1,2,3	pan object
=	1,2,3	zoom in
-	1,2,3	zoom out
x,y,z	3	rotate counterclockwise about object axis
X,Y,Z	3	rotate clockwise about object axis

## Saving TIFF files

If TIFF files are enabled, images may be saved by either the user pressing 'T' or by the program using the command `gl2SetKeyPush('T')`. Using the defaults, the first file saved is called "OpenGL001.tif", the second is "OpenGL002.tif", and so on up to "OpenGL999.tif", after which no more files are saved. The reason for not continuing indefinitely is that it is easy to accidentally save vast numbers of files, which is not discovered until the hard disk is full. The defaults can be changed using the `gl2SetOptionInt` and `gl2SetOptionStr` functions. Using the former, the starting suffix number can be changed to any value and the maximum number of TIFFs can also be set to any number. Using the latter function, the root filename can be changed. To save to a subdirectory rather than the current one, use standard path notation; for example, on a Macintosh, the filename ":Tiffs:mytiff" will save with root name `mytiff` to folder `Tiffs`.

These TIFF stacks can be loaded into QuickTime Pro and saved as a QuickTime movie.

## Assorted OpenGL facts

My reference for OpenGL is the *OpenGL Superbible* (second edition) by Wright and Sweet. While they make an effort to make the topic understandable, it is nevertheless quite confusing. Here are some important OpenGL facts.

### Coordinate systems

*Window client area.* The portion of a window that is available to be drawn in, meaning everything except for title bar, scroll bars, etc. Measured in *x* and *y* pixels.

*Viewport.* The portion of the window client area that a drawing is mapped to. Measured in *x* and *y* pixels (often equal to the window client area). Specified with `glViewport` and gotten with `glGetIntegerv(GL_VIEWPORT, ...)`.

*Clipping coordinates.* A logical coordinate system for a drawing area or volume. Nothing exists outside this area or volume. Specified with `glMatrixMode(GL_PROJECTION); glLoadIdentity();` and then modified with `glOrtho, gluOrtho2D, gluPerspective.`

### **Externally accessible functions**

```
void gl2Initialize(float xlo,float xhi,float ylo,float yhi,float zlo,float zhi);
```

This allows a program to set up a generic OpenGL output window with minimal effort. Arguments are the outside dimensions of the object; the clipping volume is created so the object can be rotated to any angle without poking outside the volume (using the arrow keys; rotation with other keys can lead to the object escaping). For 1 or two dimensional rendering, enter values of 0 for the excess dimensions. This function performs the initial viewing transformation, meaning that drawings that are within the clipping volume appear in the window. For 2-D graphics note that the option for "Fix2DAspect" needs to be set before calling `gl2Initialize.`

```
void gl2glutInit(int argc,char *argv[]);
```

This just calls `glutInit` with the same arguments. It is in this library to allow testing for the OpenGL code availability before calling the `glut` routine.

```
int gl2State(int state);
```

This allows a program to set and access a state variable that indicates whether the program should be in pause mode or quit mode. Also, the user can control the state by pressing the space bar – each press toggles the state to or from the pause mode; ‘Q’ sets the state irreversibly in the quit mode. With this routine, an argument that is equal to 0 or greater sets that state to that value, while a negative argument has no effect. Either way, the current state is returned. A state of 0 indicates normal operation, a state of 1 indicates pause mode, and a state of 2 indicates quit mode.

```
float gl2GetNumber(char *variable);
```

This allows a program to access all of the numeric global variables in the `opengl2` library. Enter a string with the exact name of the variable, listed below in the section on “Internal variables and functions,” and the value of that variable will be returned. If the variable is an integer type, its value is cast to a float. Non-numeric variables (*e.g.* strings) are not available with this command.

```
int gl2SetOptionInt(char *option,int value);
```

Sets integer options for the `opengl2` library. Use a negative number for value to not change the current value of the option. The current or new value of the option is returned. If option is “Fix2DAspect”, values of 0 or 1 are allowed, using value. If it is 0, which is the default, the edges of the window are the ones requested with `gl2Initialize`, regardless of window sizing; otherwise, scaling is the same on the *x* and *y* axes to eliminate object distortion. If option is “TiffNumber” it sets the initial number of the saved TIFF files to the value given; 1 is default. If option is “TiffNumMax” it sets the maximum number of the saved TIFF files to the value given; 999 is default. If option is “Dimension” it returns the plot dimension, although it does not allow the value to be changed.

```
float gl2SetOptionFlt(char *option,float value);
```

Sets float options for the opengl2 library. Use a negative number for value to not change the current value of the option. The current or new value of the option is returned. If option is "RotateAngle", it sets the object rotation angle in degrees; 5 is the default value.

char \*gl2SetOptionStr(char \*option, char \*value);

Sets string options for the opengl2 library. Currently, the only option available is "TiffName", which is entered as a string and can be set to any string using value. Tiff files are saved using this file name. The name should not include the number suffix or the ".tif" ending. The default is "OpenGL". Enter NULL for value for this function to return a pointer to the string; this is a pointer to the original string so it should not be changed.

void gl2SetKeyPush(unsigned char key);

This is a program accessible function that makes the graphics display act exactly as though a key had been pushed by the user. key should be equal to the character that the user would select. For the arrows (rotation), key should be 'u' for the up-arrow, 'd' for down, 'l' for left, and 'r' for right, and the corresponding uppercase characters for the shift-arrow commands (panning).

void gl2SetColor(char c);

Sets the current drawing color using a simple character for input, making this both crude and convenient. Upper and lower case letters yield the same color. Numbers and '+' and '-' symbols use the resistor convention.

#### Color codes

A	aqua	K,0	black	U	ultraviolet
B,6	blue	L	lime	V,7	violet
C	cyan	M	magenta	W,9	white
D	dark red	N	navy	X	random
E,8	grey	O,3	orange	Y,4	yellow
F	fuchsia	P	purple	Z	random
G	green	Q	quartz	1	brown
H,5	hunter green	R,2	red	-	silver
I	indigo	S	sienna	+	gold
J	olive	T	teal		

float gl2FindRotate(float \*vect1, float \*vect2, float \*axis);

This finds the rotation axis and angle that carries normalized vector vect1 into normalized vector vect2. Both of these must be pre-normalized. The angle, in degrees, is returned, as is the non-normalized rotation axis vector in axis. If vect1 and vect2 are parallel or anti-parallel, the first axis choice is perpendicular to vect1 and the x-axis (which fails if they are parallel) and the second axis choice is perpendicular to vect1 and the y-axis.

void gl2DrawBox(float \*pt1, float \*pt2, int dim);

This draws an axis-aligned wire frame box with low point pt1 and high point pt2, both of which must be 3-dimensional vectors for all dim values. dim is the box dimensionality. This function does not worry about winding directions. These

boxes are drawn with `GL_LINES`, `GL_LINE_LOOP`, and `GL_LINE_STRIP`, so the OpenGL line modifying commands are the ones to use to control how these boxes look.

- `void gl2DrawGrid(float *pt1, float *pt2, int *n, int dim);`  
This draws an axis-aligned wire frame grid with low point `pt1` and high point `pt2`, both of which must be 3-dimensional vectors for all `dim` values. `dim` is the grid dimensionality. The grid has  $n[d]$  partitions on the  $d$ 'th dimension, where `n` is a `dim`-dimensional vector and each  $n[d]$  value must be at least 1. The grid is drawn with `GL_POINTS` (1-D) or `GL_LINES` so the OpenGL line modifying commands are the ones to use to control how it looks.
- `void gl2DrawCircle(float *cent, float radius, int slices, char style);`  
This draws a flat circle centered at `cent`, with radius `radius`. The edge is not curved but a series of slices straight lines. The circle is in the  $x,y$ -plane. The style is entered as 'f' or 'g' for a filled circle, 'e' for the circle edge, or 'v' to just show the vertices along the edge.
- `void gl2DrawArc(float *cent, float radius, float theta1, float theta2, int slices, char style);`  
This is identical to `gl2DrawCircle`, but only draws the arc from `theta1` to `theta2`. The `slices` is the slices for the whole circle, which is rounded down as needed to make the arc an integer number of slices.
- `void gl2DrawHemisphere(float radius, int slices, int stacks, int frontin);`  
This draws a hemisphere in the positive  $z$  space, with its equator, which is its edge, on the  $x,y$ -plane. The pole is on the positive  $z$ -axis. Its radius is `radius`, it is subdivided into `slices` segments around the  $z$ -axis (longitude lines), and `stacks` segments along the  $z$ -axis (latitude lines). The hemisphere is drawn with `GL_QUAD_STRIP` and `GL_TRIANGLE_FAN`, so its appearance can be altered with OpenGL polygon commands (e.g. `glPolygonMode` and `glMaterial`). If `frontin` is 1, the front face is on the inside, and 0 means that it's on the outside.
- `void gl2DrawCylinder(float baseRadius, float topRadius, float height, int slices, int stacks, int frontin);`  
This draws a cylinder, frustum, or cone. It is nearly identical to the OpenGL quadrics cylinder, but easier to use. The cylinder is concentric with the  $z$ -axis with its base on the  $x,y$ -plane. The base has radius `baseRadius`, the top has radius `topRadius`, the cylinder height is `height`, and it is divided into `slices` segments around the  $z$ -axis and `stacks` segments along the  $z$ -axis. The cylinder is drawn with `GL_QUAD_STRIP`, so its appearance can be altered with OpenGL polygon commands. As elsewhere, `frontin` should be set to 1 for the front on the inside and 0 for the front on the outside.
- `void gl2DrawSphere(float radius, int slices, int stacks, int frontin);`  
This draws a sphere. It is nearly identical to the OpenGL `glutWireSphere` and `glutSolidSphere` but is slightly easier to use. The sphere is centered at the origin and has radius `radius`. It is divided into `slices` segments around the  $z$ -axis and `stacks` segments along the  $z$ -axis. It is drawn with `GL_QUAD_STRIP` and `GL_TRIANGLE_FAN` so its appearance can be altered with OpenGL polygon commands. If `frontin` is 1, the front face is on the inside, and 0 means that it's on the outside.
- `void gl2DrawEcoli(float radius, float length, int slices, int stacks, int frontin);`

This draws an *E. coli* shape, which is a cylinder capped at both ends by a hemisphere. Its axis is parallel to the  $z$ -axis and its mid-plane is the  $x,y$ -plane. It has radius *radius*, total length (including both endcaps) *length*, and it has *slices* segments around the  $z$ -axis and *stacks* total segments along the  $z$ -axis. The shape is drawn with `GL_QUAD_STRIP` and `GL_TRIANGLE_FAN`, so its appearance can be altered with OpenGL polygon commands.

```
void gl2PlotData(float *xdata, float *ydata, int nx, int nycol, char *style);
```

This is a convenient way to plot simple  $x,y$  data. *xdata* should be a one-column wide array of  $x$ -values of size *nx*. *ydata* gives the  $y$ -values for multiple data series, of size *nx* rows by *nycol* columns. The *style* string is passed in with three characters per data series. The first character is ' ' to indicate that the series should not be plotted, '-' for lines, '.' for dots, or 'h' for a histogram (which isn't written correctly yet). The second character is a number to give the boldness of the line or symbol, from 0 to 9. The third character is the color of the data series, using the codes shown above for `gl2SetColor`. Note that there is no check on the length or correctness of the *style* string.

```
void gl2PlotPts(float **data, int *ser, int nser, int npts, float **color, float *size, char style);
```

This is similar to `gl2PlotData`, but is for a 3-D scatter plot. Here, *data* is indexed as *data[row][dimension]*, where *row* is the point number and *dimension* is an index that varies from 0 to 2 for the  $x$ ,  $y$ , and  $z$  components of the point. *ser* is indexed as *ser[row]* and gives the series number (0 to *nser*-1) for each data point, *nser* is the number of data series, and *npts* is the total number of data points. *color* is indexed as *color[series][RGB]* and should be between 0 and 1 for each RGB value and *size[series]* is a float for the point size. *style* is ' ' for plot nothing, '.' for plot dots, '-' for plot lines, and 's' for plot spheres.

```
void gl2PlotSurf(float *xdata, float *ydata, float **zdata, int nx, int ny, int nz, char *style);
```

This plots data in three dimensions. *xdata* and *ydata* are the lists of  $x$  and  $y$  values, which are the independent coordinates, and have sizes *nx* and *ny*. *zdata* is the dependent variable. The first array of *zdata* is the  $x,y$  coordinate and the second array is the series number, so an  $x,y,z$  coordinate is

```
x=xdata[ix];
y=ydata[iy];
z=zdata[iy*nx+ix][iz]
```

Here, *ix* is the index on  $x$ , *iy* is the index on  $y$ , and *iz* is the series number. The first letter of the *style* string tells what the plot type is, where the only letter currently defined is 's' for shading. Using this option, only the first 64 sets of  $z$ -data are plotted; all the rest are ignored. Subsequent letters are the color codes for the respective data series. Colors for multiple series are added together. Note that there is no check on the length or correctness of *style*. Boxes around points extend to the centers between points. For example, suppose *xdata*={0,1,2,4}; then, box intervals on  $x$  are (-0.5,0.5), (0.5,1.5), (1.5,3), and (3,5).

## Internal variables and functions

```
void ChangeSize(int w,int h);
void KeyPush(unsigned char key,int x,int y);
void SpecialKeyPush2(unsigned char key,int x,int y);
void SpecialKeyPush(int key,int x,int y);
#ifdef _TIFFIO_
int WriteTIFF(char *filename,char *description,int x,int y,int width,int
             height,int compression);
#endif

GLfloat ClipSize,ClipMidx,ClipMidy,ClipMidz;
GLfloat ClipLeft,ClipRight,ClipBot,ClipTop,ClipBack,ClipFront;
GLfloat FieldOfView,Near,Aspect;
GLfloat PixWide,PixHigh;
int GL2PauseState,Dimension;
GLfloat Xtrans,Ytrans;
int Fix2DAspect=0;
char TiffName[STRCHAR];
int TiffNumber=1;
int TiffNumMax=999;
GLfloat RotateAngle=5.0;
```

## Variables

ClipSize	length of edge of clipping cube, sized so object can spin.
ClipMidx	middle of object on $x$ axis.
ClipMidy	middle of object on $y$ axis.
ClipMidz	middle of object on $z$ axis.
ClipLeft	left edge of clipping cube.
ClipRight	right edge of clipping cube.
ClipBot	bottom edge of clipping cube.
ClipTop	top edge of clipping cube.
ClipBack	back edge of clipping cube ( $z$ more negative; away from viewer).
ClipFront	front edge of clipping cube ( $z$ less negative; close to viewer).
FieldOfView	field of view on $y$ direction for 3 dimensional, in degrees.
Near	distance from viewer to front side of clipping cube.
Aspect	aspect ratio of window.
PixWide	width of window in pixels.
PixHigh	height of window in pixels.
GL2PauseState	pause state, with 0 for continue, 1 for pause, 2 for quit.
Dimension	dimensionality of object, from 1 to 3.
Xtrans	translation on $x$ -axis.
Ytrans	translation on $y$ -axis.
Fix2DAspect	option that fixes the aspect ratio in 2D to uniform if turned on; default: 0
TiffName	option for the name of a saved TIFF picture; default: "OpenGL"

TiffNumber option for the suffix number of the first saved TIFF; default: 001  
TiffNumMax option for the maximum suffix number of saved TIFFs; default: 999  
RotateAngle option for rotation angle with key manipulations; default: 5 degrees

## Functions

void ChangeSize(int w,int h);

ChangeSize is a call-back function. It is called when the user changes the window size or shape, as well as upon initialization. The function takes in the current window width and height in pixels, in w and h, respectively. Using those, it sets the clipping volume in such a way that changing the window shape does not distort the image.

KeyPush is a call-back function which is called when a key is pressed (but not arrows, etc.) It is set up to rotate the object about the object axes, based on user key pushes, as well as to zoom in and out and reset to a default view.

SpecialKeyPush2 rotates the object about the viewing axes and translates it, based on the following key values: 'u', 'd', 'l' and 'r' rotate up, down, left, and right and 'U', 'D', 'L', and 'R' translate the object up, down, left, and right. The rotation angle is RotateAngle.

SpecialKeyPush is a call-back function which is called when a special key is pressed (arrows, etc.) It converts the key presses to letter symbols and rotates or translates the object by calling SpecialKeyPush2.

WriteTIFF

This saves the current contents of the OpenGL window to disk as a TIFF file. It is saved to the same folder as the application and is called "OpenGL###.tif", where the ### starts at 001 and is incremented. This function was copied nearly verbatim from the short demonstration program called writetiff.c that was written by Mark Kilgard and copyrighted in 1997. It returns 1 if it fails, 0 if it succeeds, or 2 if the libtiff library is not available. Usually enter -1 for compression, which gets replaced with the libtiff value for COMPRESSION\_PACKBITS.

## **Bugs and desired improvements**

gl2Initialize doesn't seem to work with ymax<0.000005.

gl2SetOption should have a lot more options that can be set. For example, it should be possible to modify the viewing boundaries, or allow the calling program to rotate the object or pan around, rather than just having the user do that with keystrokes.

gl2PlotData needs work to get the histogram part functioning. Also, there should be a check on the style string.

gl2PlotSurf should be able to plot surfaces in 3D or in other ways. There should be a check on the style string.

Should change tiff filename suffix numbers to have the proper number of digits, using TiffNumMax.

