VOGL

A very ordinary GL Like Library

Documentation

README:

The directories in this directory contain the source for the VOGL library and the library hershey which is for manipulating the Hershey font data.

They are as follows:

docs	contains the documentation such as there is.
drivers	contains the source to a variety of device drivers, currently restricted to: postcript, sun workstation, apollo workstations, X11 (R2, R3 & R4), tektronix (401x), hpgl, dxy, and the ibm pc cards: hercules mono, cga, ega, vga and sigma.
examples	contains some C and FORTRAN programs useful both in testing and (hopefully) learning how to use it. There are subdirectories xview,xt and sunview showing how to use an x toolkit (or sunview) with VOGL.
hershey	contains the source for the hershey library plus the source for generating fonts and the hershey data for the occidental and oriental character sets. Note: as with VOGL this library is callable in C and FORTRAN.
src	contains the source for the C VOGL interface, and the source for the FORTRAN interfaces where available.

If you have a real SGI machine with GL on it, you can simply compile the Hershey library and the examples with "make -f Makefile.sgi". Otherwise, you will have to edit the Makefile to set various options for your machine and type "make".

VOGL is a device portable graphics library that tries to be Silicon Graphics Iris GL compatible. Our intention is that any VOGL program will compile unchanged on a machine running SGI GL (the examples do). VOGL is based entirely on our other graphics library VOGLE. While we still regard VOGLE as our main library (it is, and probably will be for some time to come, the one that gets the most use around here), we will gratefully accept any bug fixes or enhancements. As always suggestions are also welcome.

This software may be used for any purpose commercial or otherwise. It is offered without any guarantee as to its suitability for any purpose or as to the sanity of its writers. We do ask that the source is passed on to anyone that requests a copy, and that people who get copies don't go round claiming they wrote it (that is why this one has a copyright notice in it, see file COPYRIGHT).

Although VOGL is free we will drink any quantity of Beer you send to us.

Regards,

Eric H. Echidna

Snail mail correspondance and alcoholic beverages should be directed to:

The Software Support Programmer
Department Of Engineering Computer Resources
Faculty Of Engineering
University Of Melbourne Vic 3052
Australia

email to

echidna@munnari.OZ.AU
echidna@ecr.mu.OZ.AU
echidna@gondwana.ecr.mu.OZ.AU

Contents

NAME

VOGL - A very ordinary GL Like Library.

DESCRIPTION

VOGL is a library of C routines which try to allow a programmer to write programs which can be moved to machines which have the Silicon Graphics GL library on them. It is based entirely on the VOGLE graphics library, and as a result can handle circles, curves, arcs, patches, and polygons in a device independent fashion. Simple hidden line removal is also available via polygon backfacing. Access to hardware text and double buffering of drawings depends on the driver. There is also a FORTRAN interface but as it goes through the C routines FORTRAN users are warned that arrays are in row-column order in C. Both the long FORTRAN names and the shortened six character names are supported. People interested in using software text should see the hershey library, HERSHEY(3).

Some routines are only available in VOGL. If you include them in programs it is advisable to put #ifdef VOGL ... #endif around them. The constant VOGL is defined whenever a VOGL header file is included.

It should be noted that there are a number of routines that take the type Angle for some of their parameters. All angles specified this way are actually Integer Tenths Of Degrees. (Don't ask!)

Include files.

There are two include files provided with vogl: vogl.h and vodevice.h. The file vogl.h has the type definitions and function interfaces, ideally it is included where you would include gl.h on an SGI. The file vodevice.h has the devices in it, and it is included where you would include device.h on an SGI.

The following is a brief summary of the VOGL subroutines.

Using X toolkits and Sunview

For X11 and Sunview based applications, it is posible for VOGL to use a window that is supplied by that application's toolkit. Under these circumstances, the toolkit is is responsible for handling of all input events, and VOGL simply draws into the supplied window. These calls are only available from C. Also see the directories examples/xt, examples/xview and examples/sunview.

For X based toolkits the following two calls may be used:

vo_xt_window(display, xwin, width, height)
Tells VOGL to use the supplied window xwin

```
vo_xt_window(display, xwin, width, height)
     Display   *display;
     Window    xwin;
     int  width, height;
```

This routine should be called before calling "ginit()".

vo_xt_win_size(width, height)
Tells VOGL that the supplied window has changed size.

```
vo_xt_win_size(width, height)
     int  width, height;
```

For sunview based applications the following two calls may be used:

vo_sunview_canvas(canvas, width, height)
Tells VOGL to use the supplied sunview canvas canvas

```
vo_sunview_canvas(canvas, width, height)
          Canvas    canvas;
          int  width, height;
```

This routine should be called before calling "ginit()".

vo_sunview_canvas_size(width, height)
Tells VOGL that the supplied canvas has changed size.

```
vo_sunview_canvas_size(width, height)
     int  width, height;
```

Device routines.

vinit(device)
Tell VOGL what the device is. This routine needs to be called if the environment variable VDEVICE isn't set, or if the value in VDEVICE is not to be used.

```
Fortran:
     subroutine vinit(device, len)
     character *(*) device
     integer len

C:
```

```
          vinit(device);
          char     *device;

     Note 1 :- Current available devices are:
              tek - tektronix 4010 and compatibles
              hpgl - HP Graphics language and compatibles
              dxy - roland DXY plotter language
              postscript - monochrome postscript devices
             ppostscript - monochrome postscript devices (portrait mode)
              cps - colour postscript devices
              pcps - colour postscript devices (portrait mode)
              grx - the GRX library that is part of DJGPP.
                    - (little tested)
              sun - Sun workstations running sunview
              X11 - X windows (SUN's Openwindows etc etc)
              decX11 - the decstation (old) window manager
                    This is only included in case you need it.
              apollo - Apollo workstations
              NeXT   - NeXTStep
              hercules - IBM PC hercules graphics card
              cga - IBM PC cga graphics card
              ega - IBM PC ega graphics card
              vga - IBM PC vga graphics card
              sigma - IBM PC sigma graphics card.
              mswin - MS-windoze (little tested).

              Sun, X11, decX11, apollo, hercules, cga
              and ega support double buffering.

     Note 2 :- If device is a NULL or a null string the value
          of the environment variable "VDEVICE" is taken as the
          device type to be opened.

     Note 3 :- after init it is wise to explicitly
          clear the screen.

     e.g.: in C
          color(BLACK);
          clear();

     or    in Fortran
          call color(BLACK)
          call clear
```

ginit()

Open the graphics device and do the basic initialisation. This routine is marked for obsolescence. The routine winopen (see below) should be used instead. Note:

this automatically causes a REDRAW event to appear as the first event in the event queue.

```
Fortran:
     subroutine ginit

C:
     ginit()
```

winopen(title)

Open the graphics device and do the basic initialisation. This routine should be used instead of ginit. Note: this automatically causes a REDRAW event to appear as the first event in the event queue.

```
Fortran:
     subroutine winopen(title, len)
     character*(*) title
     integer len

C:
     winopen(title)
          char *title;
```

gexit()

Reset the window/terminal (must be the last VOGL routine called)

```
Fortran:
     subroutine gexit

C:
     gexit()
```

voutput(path)

Redirect output from *next* ginit to file given by path. This routine only applies to devices drivers that write to stdout e.g. postscript and hpgl.

```
Fortran:
     subroutine voutput(path, len)
     character*(*) path
     integer len

C:
     voutput(path)
          char *path;
```

vnewdev(device)

Reinitialize VOGL to use a new device without changing attributes, viewport etc. (eg. window and viewport specifications)

```
Fortran:
     subroutine vnewdev(device, len)
     character *(*) device
     integer len

C:
     vnewdev(device)
          char *device;
```

getplanes() Returns the number of bit planes (or color planes) for a particular device. The number of colors displayable by the device is then 2**(nplanes-1)

```
Fortran:
     integer function  getplanes()

C:
     long
     getplanes()
```

Routines for controlling flushing or syncronisation

On some devices (particularly X11) considerable speedups in display can be achieved by not flushing each graphics primitive call to the actual display until necessary. VOGL automatically delays flushing under in following cases:

- Within a callobj() call.
- Within curves and patches.
- Within bgn*/end* calls.
- When double buffering (the flush is only done withing swapbuffers).

There are two user routines (which are NOT GL compatible) that can be used to control flushing.

vsetflush(yesno)

Set global flushing status. If yesno = 0 (.false.) then don't do any flushing (except in swapbuffers(), or vflush()). If yesno = 1 (.true.) then do the flushing as described above.

```
Fortran:
     subroutine vsetflush(yesno)
     logical yesno

C:
     void
```

```
vsetflush(yesno)
     int  yesno;
```

vflush()

Call the device flush or syncronisation routine. This forces a flush.

Fortran:

```
subroutine vflush
```

C:

```
void
vflush();
```

Routines For Setting Up Windows.

Some devices are basically window orientated - like sunview and X11. You can give VOGL some information about the window that it will use with these routines. These can make your code very device dependent. Both routines take arguments which are in device space. (0, 0) is the bottom left hand corner in device space. To have any effect these routines must be called before ginit or winopen. For the X11 device, an entry may be made in your .Xdefaults file of the form vogl.Geometry =150x500+550+50 (where you specify your geometry as you please).

prefposition(x1, x2, y1, y2)

Specify the preferred position of the window opened by the *next* winopen.

Fortran:

```
subroutine prefposition(x1, x2, y1, y2)
integer x1, x2, y1, y2
```

C:

```
prefposition(x1, x2, y1, y2)
     long x1, x2, y1, y2
```

prefsize(width, height)

Specify the preferred width and height of the window opened by the *next* winopen.

Fortran:

```
subroutine prefsize(width, height)
integer width, height
```

C:

```
prefsize(width, height)
     long width, height;
```

reshapeviewport

This is occasionally used in Iris GL if a REDRAW event rolls up. While VOGL is unlikely to ever provide a REDRAW event (except possibly the first event in the event queue) the call is provided for compatibility.

```
Fortran:
     subroutine reshap

C:
     reshapeviewport()
```

General Routines.

clear()

Clears the current viewport to the current colour.

```
Fortran:
     subroutine clear

C:
     clear()
```

color(col)

Set the current colour. The standard colours are as follows:

black = 0	red = 1	green = 2	yellow = 3
blue = 4	magenta = 5	cyan = 6	white = 7.

These are included in vogl.h as:

BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN and WHITE.

When using fortran these are included in fvogl.h as
BLACK, RED, GREEN, YELLOW, BLUE, MAGENT, CYAN and WHITE.

```
Fortran:
     subroutine color(col)
     integer col

C:
     color(col)
          Colorindex     col;
```

colorf(col)

Same as color only it takes a floating point argument. In Iris GL there are sometimes good reasons for using this routine over color. See the GL manual for more details.

```
Fortran:
     subroutine colorf(col)
     real col

C:
     colorf(col)
          float     col;
```

mapcolor(indx, red, green, blue)

Set the color map index indx to the color represented by (red, green, blue). If the device has no color map this call does nothing.

```
Fortran:
     subroutine mapcolor(indx, red, green, blue)
     integer indx, red, green, blue

C:
     mapcolor(indx, red, green, blue)
          Colorindex     indx;
          short          red, green, blue;
```

defbasis(id, mat)

Define basis number id to be the matrix mat.

```
Fortran:
     subroutine defbasis(id, mat)
     integer id
     real mat(4, 4)

C:
     defbasis(id, mat)
          short     id;
          Matrix    mat;
```

polymode(mode)

NOTE:- For this call to have any effect it must have been conditionally compiled into the library. (See polygons.c for details) Control the filling of polygons. It expects one of the following PYM_LINE, which means only the edges of the polygon will be drawn and PYM_FILL which means fill the polygon (the default). PYM_POINT and PYM_HOLLOW are also recognised but they don't behave quite as they would with SGI GL.

Also note that in Fortran the corresponding constants are truncated to PYM_LI, PYM_FI, PYM_PO and PYM_HO respectivly. These appear in fvogl.h.

```
Fortran:
     subroutine polymode(mode)
     integer mode

C:
     polymode(mode)
          long mode;
```

The Device Queue and Valuator Routines.

The available devices are defined in the header files vodevice.h and for FORTRAN fvodevice.h

qdevice(dev)

Enable a device. Note: in VOGL the queue is of length 1.

```
Fortran:
     subroutine qdevice(dev)
     integer dev

C:
     qdevice(dev)
          Device    dev;
```

unqdevice(dev)

Disable a device.

```
Fortran:
     subroutine qdevice(dev)
     integer dev

C:
     qdevice(dev)
          Device    dev;
```

qread(data)

Read an event from the device queue. This routines blocks until something happens. Note: it is important to have called qdevice before doing this.

```
Fortran:
     integer function qread(data)
     integer*2 data

C:
     long qread(data)
          short     *data;
```

isqueued(dev)

Check to see if device dev is enabled for queueing.

```
Fortran:
     logical function isqueued(dev)
     integer dev

C:
     Boolean isqueued(dev)
          short     *dev;
```

qtest()

Check if there is anything in the queue. Note: in VOGL the queue is only 1 entry deep.

```
Fortran:
     logical function qtest

C:
     Boolean qtest()
```

qreset()

Reset the device queue. This will get rid of any pending events.

```
Fortran:
     subroutine qreset

C:
     qreset()
```

getbutton(dev)

Returns the up (0) or down (1) state of a button.

```
Fortran:
     logical function getbutton(dev)
     integer dev

C:
     Boolean getbutton(dev)
          Device    dev;
```

getvaluator(dev)

Return the current value of the valuator. Currently the only valuators supported are MOUSEX and MOUSEY.

```
Fortran:
     integer function getvaluator(dev)
```

```
integer dev
```

C:

```
long getvaluator(dev)
     Device    dev;
```

Viewport Routines.

viewport(left, right, bottom, top)

Specify which part of the screen to draw in. Left, right, bottom, and top are integer values in screen coordinates.

Fortran:

```
subroutine viewport(left, right, bottom, top)
integer left, right, bottom, top
```

C:

```
viewport(left, right, bottom, top)
     Screencoord      left, right, bottom, top;
```

pushviewport()

Save current viewport on the viewport stack.

Fortran:

```
subroutine pushviewport
```

C:

```
pushviewport()
```

popviewport()

Retrieve last pushed viewport.

Fortran:

```
subroutine popviewport
```

C:

```
popviewport()
```

getviewport(left, right, bottom, top)

Returns the left, right, bottom and top limits of the current viewport in screen coordinates.

Fortran:

```
subroutine getviewport(left, right, bottom, top)
integer*2 left, right, bottom, top
```

C:

```
getviewport(left, right, bottom, top)
```

```
Screencoord      *left, *right, *bottom, *top;
```

Attribute Stack Routines.

The attribute stack contains details such as current color, current line style and width, and the current font number. If you need to prevent object calls form changing these, use pushattributes before the call and popattributes after.

pushattributes()

Save the current attributes on the attribute stack.

```
Fortran:
     subroutine pushattributes

C:
     pushattributes()
```

popattributes()

Restore the attributes to what they were at the last pushattribute().

```
Fortran:
     subroutine popattributes

C:
     popattributes()
```

Projection Routines.

All the projection routines define a new transformation matrix, and consequently the world units. Parallel projections are defined by ortho or ortho2. Perspective projections can be defined by perspective and window. Note the types Angle, etc, are defined in vogl.h. Remember angles are in tenths of degrees.

ortho(left, right, bottom, top, near, far)

Define x (left, right), y (bottom, top), and z (near, far) clipping planes. The near and far clipping planes are actually specified as distances along the line of sight. These distances can also be negative. The actual location of the clipping planes is z = -near_d and z = -far_d.

```
Fortran:
     subroutine ortho(left, right, bottom, top, near_d, far_d)
     real left, right, bottom, top, near_d, far_d

C:
     ortho(left, right, bottom, top, near_d, far_d)
          Coord     left, right, bottom, top, near_d, far_d;
```

ortho2(left, right, bottom, top)

Define x (left, right), and y (bottom, top) clipping planes.

```
Fortran:
     subroutine ortho2(left, right, bottom, top)
     real left, right, bottom, top

C:
     ortho2(left, right, bottom, top)
          float     left, right, bottom, top;
```

perspective(fov, aspect, near, far)

Specify a perspective viewing pyramid in world coordinates by giving a field of view, aspect ratio and the distance from the eye of the near and far clipping plane.

```
Fortran:
     subroutine perspective(fov, aspect, near, far)
     integer fov
     real aspect, near, far

C:
     perspective(fov, aspect, near, far)
          Angle     fov;
          float     aspect;
          Coord     near, far;
```

window(left, right, bot, top, near, far)

Specify a perspective viewing pyramid in world coordinates by giving the rectangle closest to the eye (ie. at the near clipping plane) and the distances to the near and far clipping planes.

```
Fortran:
     subroutine window(left, right, bot, top, near, far)
     real left, right, bot, top, near, far

C:
     window(left, right, bot, top, near, far)
          float     left, right, bot, top, near, far;
```

Matrix Stack Routines.

pushmatrix()

Save the current transformation matrix on the matrix stack.

```
Fortran:
     subroutine pushmatrix
```

```
          C:
               pushmatrix()
```

popmatrix()

Retrieve the last matrix pushed and make it the current transformation matrix.

```
          Fortran:
               subroutine popmatrix

          C:
               popmatrix()
```

Viewpoint Routines.

Viewpoint routines alter the current tranformation matrix.

polarview(dist, azim, inc, twist)

Specify the viewer's position in polar coordinates by giving the distance from the viewpoint to the world origin, the azimuthal angle in the x-y plane, measured from the y-axis, the incidence angle in the y-z plane, measured from the z-axis, and the twist angle about the line of sight.

```
          Fortran:
               subroutine polarview(dist, azim, inc, twist)
               real dist
               integer azim, inc, twist

          C:
               polarview(dist, azim, inc, twist)
                    Coord     dist;
                    Angle     azim, inc, twist;
```

lookat(vx, vy, vz, px, py, pz, twist)

Specify the viewer's position by giving a viewpoint and a reference point in world coordinates. A twist about the line of sight may also be given.

```
          Fortran:
               subroutine lookat(vx, vy, vz, px, py, pz, twist)
               real vx, vy, vz, px, py, pz
               integer twist

          C:
               lookat(vx, vy, vz, px, py, pz, twist)
                    float     vx, vy, vz, px, py, pz;
                    Angle     twist;
```

Move Routines.

There are variations on all these routines that end in 's' and also end in 'i'. In the case of the 's' variations they take arguments of type Scoord in C and integer*2 in FORTRAN. In the case of the 'i' variations they take arguments of type Icoord in C and integer in FORTRAN.

move(x, y, z)

Move current graphics position to (x, y, z). (x, y, z) is a point in world coordinates.

```
Fortran:
     subroutine move(x, y, z)
     real x, y, z

C:
     move(x, y, z)
          Coord     x, y, z;
```

rmv(deltax, deltay, deltaz)

Relative move. deltax, deltay, and deltaz are offsets in world units.

```
Fortran:
     subroutine rmv(deltax, deltay, deltaz)
     real deltax, deltay, deltaz

C:
     rmv(deltax, deltay, deltaz)
          Coord   deltax, deltay, deltaz;
```

move2(x, y)

Move graphics position to point (x, y). (x, y) is a point in world coordinates.

```
Fortran:
     subroutine move2(x, y)
     real x, y

C:
     move2(x, y)
          Coord     x, y;
```

rmv2(deltax, deltay)

Relative move2. deltax and deltay are offsets in world units.

```
Fortran:
     subroutine rmv2(deltax, deltay)
```

```
          real deltax, deltay

     C:
          rmv2(deltax, deltay)
               Coord     deltax, deltay;
```

Line routines.

These routines set the line style and line width if the current device is capable of doing so.

deflinestyle(n, style)

Define a line style and binds it to the integer n. The line style is a bit pattern of 16 bits width.

```
     Fortran:
          subroutine deflin(n, style)
          integer   n
          integer style

     C:
          deflinestyle(n, style)
               short     n;
               Linestyle style;
```

setlinestyle(n)

Sets the current line style.

```
     Fortran:
          subroutine setlin(n)
          integer   n

     C:
          setlinestyle(n)
               short     n;
```

linewidth(n)

Sets the current line width to 'n' pixels wide.

```
     Fortran:
          subroutine linewi(n)
          integer   n

     C:
          linewidth(n)
               short     n;
```

Drawing Routines.

There are variations on all these routines that end in 's' and also end in 'i'. In the case of the 's' variations they take arguments of type Scoord in C and integer*2 in FORTRAN. In the case of the 'i' variations they take arguments of type Icoord in C and integer in FORTRAN.

draw(x, y, z)

Draw from current graphics position to (x, y, z). (x, y, z) is a point in world coordinates.

```
Fortran:
     subroutine draw(x, y, z)
     real x, y, z

C:
     draw(x, y, z)
          Coord     x, y, z;
```

rdr(deltax, deltay, deltaz)

Relative draw. deltax, deltay, and deltaz are offsets in world units.

```
Fortran:
     subroutine rdr(deltax, deltay, deltaz)
     real deltax, deltay, deltaz

C:
     rdr(deltax, deltay, deltaz)
          Coord   deltax, deltay, deltaz;
```

draw2(x, y)

Draw from current graphics position to point (x, y). (x, y) is a point in world coordinates.

```
Fortran:
     subroutine draw2(x, y)
     real x, y

C:
     draw2(x, y)
          Coord     x, y;
```

rdr2(deltax, deltay)

Relative draw2. deltax and deltay are offsets in world units.

```
Fortran:
     subroutine rdr2(deltax, deltay)
```

```
          real deltax, deltay

     C:
          rdr2(deltax, deltay)
               Coord   deltax, deltay;
```

Vertex calls.

There are calls which we term 'vertex calls' which simply specify a point in 4D, 3D or 2D. These calls take an array which specifies the coordinates of the point. The interpretation of these points is described below.

v4d(v) Specify a vertex(point) in 4D using double precision numbers.

```
Fortran:
     subroutine v4d(v)
     real *8 v(4)

C:
     v4d(v)
          double v[4];
```

v4f(v) Specify a vertex(point) in 4D using single precision floating point numbers.

```
Fortran:
     subroutine v4f(v)
     real v(4)

C:
     v4f(v)
          float v[4];
```

v4i(v) Specify a vertex(point) in 4D using integer numbers

```
Fortran:
     subroutine v4i(v)
     integer v(4)

C:
     v4i(v)
          long v[4];
```

v4s(v) Specify a vertex(point) in 4D using short integer numbers

```
Fortran:
     subroutine v4s(v)
     integer *2  v(4)

C:
     v4s(v)
          short v[4];
```

There are also equivalent calls for 3D points (v3d, v3f, v3i, v3s) and 2D points (v2d, v2f, v2i, v2s). The only difference is the number of elements that each vertex needs to be specified. It should also be noted the the different data types (ie. double, float, long and short) are merely different ways of representing the same basic coordinate data (calling v3s with v[] = {100,200,200} is the same as calling v3f with v[] = {100.0, 200.0, 200.0}).

The way these points are interpreted depends on what mode has be set up with one of the calls bgnpoint, bgnline, bgnclosedline or bgnpolygon. The bgnpoint call specifies that the next series of vertex calls are specifying a chain of points (dots) to be drawn. A bgnpoint is terminated with a endpoint call.

```
Fortran:
     subroutine bgnpoint

C:
     bgnpoint()

Fortran:
     subroutine endpoint

C:
     endpoint()
```

The bgnline call specifies that the next series of vertex calls are specifying the points on a polyline. A bgnline is terminated with a endline call.

```
Fortran:
     subroutine bgnline

C:
     bgnline()

Fortran:
```

```
subroutine endline
```

C:

```
endline()
```

The bgnclosedline call is similar to the bgnline except that when endclosedline is called the first point given (ie. the one first after the bgnclosedline call) is joined to the last point given (ie. the one just before the endclosedline call).

Fortran:

```
subroutine bgncloseline
```

C:

```
bgnclosedline()
```

Fortran:

```
subroutine endclosedline
```

C:

```
endclosedline()
```

The bgnpolygon call specifies that the next series of vertex calls are defining a polygon. When endpolygon is called, the polygon is closed and filled (or drawn as an outline depending on the mode that has been set with the polymode call if this call has been compilied into the library.

Fortran:

```
subroutine bgnpolygon
```

C:

```
bgnpolygon()
```

Fortran:

```
subroutine endpolygon
```

C:

```
endpolygon()
```

Arcs and Circles.

There are variations on all these routines that end in 's' and also end in 'i'. In the case of the 's' variations they take arguments of type Scoord in C and integer*2 in FORTRAN. In the case of the 'i' variations they take arguments of type Icoord in C and integer in FORTRAN.

circleprecision(nsegs)
Set the number of line segments making up a circle. Default is currently 32. The number of segments in an arc is calculated from nsegs according the span of the arc. This routine is only available in VOGL.

```
Fortran:
     subroutine circleprecision(nsegs)
     integer   nsegs
C:
     circleprecision(nsegs)
          int  nsegs;
```

arc(x, y, radius, startang, endang)
Draw an arc. x, y, and radius are values in world units.

```
Fortran:
     subroutine arc(x, y, radius, startang, endang)
     real x, y, radius;
     integer startang, endang;
C:
     arc(x, y, radius, startang, endang)
          Coord  x, y, radius;
          Angle  startang, endang;
```

arcf(x, y, radius, startang, endang)
Draw a filled arc. x, y, and radius are values in world units. (How the filling is done may be changed by calling polymode , if this call has been compilied into the library).

```
Fortran:
     subroutine arcf(x, y, radius, startang, endang)
     real x, y, radius;
     integer startang, endang;
C:
     arcf(x, y, radius, startang, endang)
          Coord  x, y, radius;
          Angle  startang, endang;
```

circ(x, y, radius)
Draw a circle. x, y, and radius are values in world units.

```
Fortran:
```

```
          subroutine circ(x, y, radius)
          real x, y, radius
     C:
          circ(x, y, radius)
               Coord     x, y, radius;
```

circf(x, y, radius)

Draw a filled circle. x, y, and radius are values in world units. How the filling is done may be changed by calling polymode.

```
     Fortran:
          subroutine circf(x, y, radius)
          real x, y, radius
     C:
          circf(x, y, radius)
               Coord     x, y, radius;
```

Curve Routines.

curvebasis(id)

Set the basis matrix for a curve to the matrix referenced by id. The matrix and it's id are tied together with a call to defbasis.

```
     Fortran:
          subroutine curvebasis(id)
          integer id
     C:
          curvebasis(id)
               short     id;
```

curveprecision(nsegs)

Define the number of line segments used to draw a curve.

```
     Fortran:
          subroutine curveprecision(nsegs)
          integer nsegs

     C:
          curveprecision(nsegs)
               short     nsegs;
```

rcrv(geom)

Draw a rational curve.

```
     Fortran:
          subroutine rcrv(geom)
```

```
                    real geom(4,4)
               C:
                    rcrv(geom)
                         Coord     geom[4][4];
```

rcrvn(n, geom)

Draw n - 3 rational curve segments. Note: n must be at least 4.

```
               Fortran:
                    subroutine rcrvn(n, geom)
                    integer n
                    real geom(4,n)
               C:
                    rcrvn(n, geom)
                         long n;
                         Coord     geom[][4];
```

crv(geom)

Draw a curve.

```
               Fortran:
                    subroutine crv(geom)
                    real geom(3,4)
               C:
                    crv(geom)
                         Coord     geom[4][3];
```

crvn(n, geom)

Draw n - 3 curve segments. Note: n must be at least 4.

```
               Fortran:
                    subroutine crvn(n, geom)
                    integer n
                    real geom(3,n)
               C:
                    crvn(n, geom)
                         long n;
                         Coord     geom[][3];
```

curveit(n)

Draw a curve segment by iterating the top matrix in the matrix stack as a forward difference matrix. This performs 'n' iterations.

```
               Fortran:
                    subroutine curveit(n)
                    integer n
```

```
C:
     curveit(n)
          short     n;
```

Rectangles and General Polygon Routines.

See also Vertex calls above. The way in which filled polygons (including circles and arcs) are treated depends on the mode that has been set with the polymode call.

There are variations on all these routines that end in 's' and also end in 'i'. In the case of the 's' variations they take arguments of type Scoord in C and integer*2 in FORTRAN. In the case of the 'i' variations they take arguments of type Icoord in C and integer in FORTRAN.

rect(x1, y1, x2, y2)
Draw a rectangle.

```
Fortran:
     subroutine rect(x1, y1, x2, y2)
     real x1, y1, x1, y2
C:
     rect(x1, y1, x2, y2)
          Coord     x1, y1, x2, y2;
```

rectf(x1, y1, x2, y2)
Draw a filled rectangle. (How the filling is done may be changed by calling polymode , if this call has been compilied into the library).

```
Fortran:
     subroutine rectf(x1, y1, x2, y2)
     real x1, y1, x1, y2
C:
     rectf(x1, y1, x2, y2)
          Coord     x1, y1, x2, y2;
```

poly2(n, points)
Construct a (x, y) polygon from an array of points provided by the user.

```
Fortran:
     subroutine poly2(n, points)
     integer n
     real points(2, n)
C:
     poly2(n, points)
          long n;
          Coord     points[][2];
```

polf2(n, points)

Construct a filled (x, y) polygon from an array of points provided by the user. (How the filling is done may be changed by calling polymode , if this call has been compiled into the library).

```
Fortran:
     subroutine polf2(n, points)
     integer n
     real points(2, n)
C:
     polf2(n,  points)
          long n;
          Coord     points[][2];
```

poly(n, points)

Construct a polygon from an array of points provided by the user.

```
Fortran:
     subroutine poly(n, points)
     integer n
     real points(3, n)
C:
     poly(n,  points)
          long n;
          float     points[][3];
```

polf(n, points)

Construct a filled polygon from an array of points provided by the user. (How the filling is done may be changed by calling polymode , if this call has been compiled into the library).

```
Fortran:
     subroutine polf(n, points)
     integer n
     real points(3, n)
C:
     polf(n, points)
          long n;
          Coord     points[][3];
```

backface(onoff)

Turns on culling of backfacing polygons. A polygon is backfacing if it's orientation in *screen* coords is clockwise.

```
Fortran:
```

```
subroutine backface(onoff)
logical onoff
```

C:

```
backface(onoff)
     Boolean   onoff;
```

frontface(onoff)

Turns on culling of frontfacing polygons. A polygon is frontfacing if it's orientation in *screen* coords is anticlockwise.

Fortran:

```
subroutine frontface(clockwise)
logical onoff
```

C:

```
frontface(clockwise)
     Boolean   onoff;
```

Text routines.

The original VOGLE hardware fonts "small" and "large" have the font numbers 0 and 1 respectively. The default font is 0. For X11 displays the default fonts used by the program can be overridden by placing the following defaults in the ~/.Xdefaults file:

```
vogl.smallfont: <font name>
vogl.largefont: <font name>
```

font(fontid)

Set the current font

Fortran:

```
subroutine font(fontid)
integer fontid;
```

C:

```
font(fontid)
     short     fontid;
```

cmov(x, y, z)

Change the current character position. The usual variations with the extensions 'i' and 's' also apply here.

Fortran:

```
subroutine cmov(x, y, z)
real x, y, z;
```

C:

```
cmov(x, y, z)
     Coord     x, y, z;
```

cmov2(x, y)

Change the current character position in x and y. The usual variations with the extensions 'i' and 's' also apply here.

```
Fortran:
     subroutine cmov2(x, y)
     real x, y;

C:
     cmov2(x, y)
          Coord     x, y;
```

getheight()

Return the maximum height in the current font.

```
Fortran:
     integer function getheight

C:
     long
     getheight()
```

strwidth(s)

Return the length of the string s in screen coords.

```
Fortran:
     integer function strwidth(s, n)
          character *(*) s
          integer   n;

C:
     long
     strwidth(s)
          char *s;
```

Transformation Routines

All transformations are cumulative, so if you rotate something and then do a translate you are translating relative to the rotated axes. If you need to preserve the current transformation matrix use pushmatrix(), do the drawing, and then call popmatrix() to get back where you were before.

translate(x, y, z)

Set up a translation.

```
          Fortran:
               subroutine translate(x, y, z)
               real x, y, z
          C:
               translate(x, y, z)
                    Coord     x, y, z;
```

scale(x, y, z)
Set up scaling factors in x, y, and z axis.

```
          Fortran:
               subroutine scale(x, y, z)
               real x, y, z

          C:
               scale(x, y, z)
                    Coord     x, y, z;
```

rot(angle, axis)
Set up a rotation in axis axis. Axis is one of 'x', 'y', or 'z'. The angle in this case is a real number in degrees.

```
          Fortran:
               subroutine rot(angle, axis)
               real angle
               character axis

          C:
               rot(angle, axis)
                    float     angle;
                    char axis;
```

rotate(angle, axis)
Set up a rotation in axis axis. Axis is one of 'x', 'y', or 'z', and the angle is in tenths of degrees. Makes you feel sentimental doesn't it.

```
          Fortran:
               subroutine rotate(angle, axis)
               integer angle
               character axis

          C:
               rotate(angle, axis)
                    Angle     angle;
                    char axis;
```

Patch Routines.

patchbasis(tbasisid, ubasisid)

Define the t and u basis matrix id's of a patch. It is assumed that tbasisid and ubasisid have matrices associated with them already (this is done using the defbasis call).

```
Fortran:
     subroutine patchbasis(tid, uid)
     integer tid, uid

C:
     patchbasis(tid, ubid)
          long tid, uid
```

patchprecision(tseg, useg)

Set the minimum number of line segments making up curves in a patch.

```
Fortran:
     subroutine patchprecision(tseg, useg)
     integer tseg, useg

C:
     patchprecision(tseg, useg)
          long     tseg, useg;
```

patchcurves(nt, nu)

Set the number of curves making up a patch.

```
Fortran:
     subroutine patchcurves(nt, nu)
     integer nt, nu

C:
     patchcurves(nt, nu)
          long     nt, nu;
```

rpatch(gx, gy, gz, gw)

Draws a rational patch in the current basis, according to the geometry matrices gx, gy, gz, and gw.

```
Fortran:
     subroutine rpatch(gx, gy, gz, gw)
     real  gx(4,4), gy(4,4), gz(4,4), gw(4,4)

C:
     rpatch(gx, gy, gz, gw)
         Matrix  gx, gy, gz, gw;
```

patch(gx, gy, gz)
Draws a patch in the current basis, according to the geometry matrices gx, gy, and gz.

```
Fortran:
     subroutine patch(gx, gy, gz)
     real  gx(4,4), gy(4,4), gz(4,4)

C:
     patch(gx, gy, gz)
          Matrix  gx, gy, gz;
```

Point Routines.

There are variations on all these routines that end in 's' and also end in 'i'. In the case of the 's' variations they take arguments of type Scoord in C and integer*2 in FORTRAN. In the case of the 'i' variations they take arguments of type Icoord in C and integer in FORTRAN.

pnt(x, y, z)
Draw a point at x, y, z

```
Fortran:
     subroutine pnt(x, y, z)
     real x, y, z

C:
     pnt(x, y, z)
          Coord     x, y, z;
```

pnt2(x, y)
Draw a point at x, y.

```
Fortran:
     subroutine pnt2(x, y)
     real x, y

C:
     pnt2(x, y)
          Coord     x, y;
```

Object Routines.

Objects are graphical entities created by the drawing routines called between makeobj and closeobj. Objects may be called from within other objects. When an object is created most of the calculations required by the drawing routines called within it are done up to where the calculations involve the current transformation matrix. So if you need to draw the same thing several times on the screen but in different places it is faster to use objects than to call the appropriate drawing routines each time.

makeobj(n)
Commence the object number n.

Fortran:
```
subroutine makeobj(n)
integer n
```

C:
```
makeobj(n)
     Object    n;
```

closeobj()
Close the current object.

Fortran:
```
subroutine closeobj()
```

C:
```
closeobj()
```

genobj()
Returns a unique object identifier.

Fortran:
```
integer function genobj()
```

C:
```
Object
genobj()
```

getopenobj()
Return the number of the current object.

Fortran:
```
integer function getopenobj()
```

C:
```
Object
getopenobj()
```

callobj(n)
Draw object number n.

Fortran:
```
subroutine callobj(n)
```

```
          integer n

     C:
          callobj(n)
               Object    n;
```

isobj(n)

Returns non-zero if there is an object of number n.

```
     Fortran:
          logical function isobj(n)
          integer n

     C:
          Boolean
          isobj(n)
               Object    n;
```

delobj(n)

Delete the object number n.

```
     Fortran:
          subroutine delobj(n)
          integer n

     C:
          delobj(n)
               Object    n;
```

Double Buffering.

Where possible VOGL allows for front and back buffers to enable things like animation and smooth updating of the screen. Note: it isn't possible to have backbuffer and frontbuffer true at the same time.

gconfig

With Iris GL you must call gconfig for things like doublebuffering to take effect.

```
     Fortran:
          subroutine gconfig

     C:
          gconfig()
```

doublebuffer

Flags our intention to do double buffering.

Fortran:
```
subroutine doublebuffer
```

C:
```
doublebuffer()
```

singlebuffer
Switch back to singlebuffer mode.

Fortran:
```
subroutine singlebuffer
```

C:
```
singlebuffer()
```

backbuffer(Boolean)
Make VOGL draw in the backbuffer.

Fortran:
```
subroutine backbuffer(yesno)
     logical   yesno;
```

C:
```
backbuffer(yesno)
     Boolean   yesno;
```

frontbuffer(Boolean)
Make VOGL draw in the front buffer.

Fortran:
```
subroutine frontbuffer(yesno)
     logical   yesno;
```

C:
```
frontbuffer(yesno)
     Boolean   yesno;
```

swapbuffers()
Swap the front and back buffers.

Fortran:
```
subroutine swapbuffers
```

C:
```
swapbuffers()
```

Position Routines.

getgpos(x, y, z, w)

Gets the current graphics position in world coords.

```
Fortran:
     subroutine getgpos(x, y, z, w)
     real x, y, z

C:
     getgpos(x, y, z, w)
          Coord *x, *y, *z, *w;
```

getcpos(ix, iy)

Gets the current character position in screen coords.

```
Fortran:
     subroutine getcpo(ix, iy)
     integer ix, iy

C:
     getcpos(ix, iy)
          Scoord *ix, *iy;
```

BUGS

Double buffering isn't supported on all devices.

The yobbarays may be turned on or they may be turned off.

INDEX