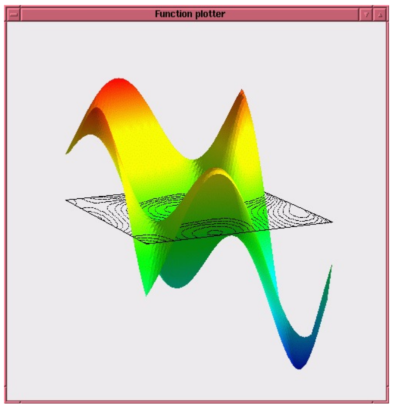
Portable Graphics from Fortran with OpenGL

William F. Mitchell
Mathematical and Computational Sciences Division
National Institute of Standards and Technology
Gaithersburg, MD 20899-8910

Fortran programmers frequently require the use of graphics, for example with scientific visualization. There have always been solutions for this requirement through various graphics libraries or language extensions. But any given library was only available on a limited number of computers, and language extensions by their very nature are compiler specific. Thus porting a program to a new computer could mean hours or days of painstakingly replacing all the graphics calls with those supported on the new system. Recently the OpenGL graphics API has been standardized by a consortium of industry leaders, and nearly all computers today support an OpenGL library. Although the OpenGL API was originally developed for C, Fortran 90 bindings have now been approved, and an implementation of the bindings, f90gl, has been ported to most Windows and Unix environments. This provides the capability of writing Fortran graphics programs that are portable across the vast majority of today's computers.

OpenGL is a powerful 3D-oriented renderer. Many graphics cards support hardware acceleration of OpenGL commands for high performance. Objects to be rendered are constructed of points, line segments and polygons positioned in two or three dimensional space, and a viewing position and direction is selected. The true power of OpenGL lies in the properties that can be attached to



objects, and the ability to specify lighting conditions to enhance the 3D appearance. Properties include color, material properties, stipple masks, and textures obtained by mapping an image file onto the object. Figure 1 contains an example of OpenGL graphics created by the plotfunc.f90 example from the f90gl distribution. An integral part of OpenGL is the OpenGL Utility Library, GLU, which contains higher level operations for view manipulation, tessellation, NURBS, and common 3D objects like spheres and cylinders. OpenGL libraries are available for nearly all computers, and come bundled with many operating systems. For other operating systems they are available for free download or as a commercial product. A free version, MESA, is available at http://www.mesa3d.org as source code. Information about OpenGL is available at http://www.opengl.org. "The OpenGL Programming Guide", commonly referred to as "the red book" was written by members of the OpenGL Architecture Review Board and discusses all OpenGL functions and their syntax.

OpenGL provides facilities for creating and rendering objects, but it does not address the actual display of the resulting image. This is necessarily dependent on the display device, for example a window system on a computer monitor. One approach to display the actual image is to use the native system commands for operations like creating windows and reading mouse events, for example the Windows API or X Windows library. However, any program using this approach will not be portable across multiple environments. To obtain portability, the OpenGL Utility Toolkit, GLUT, was created and ported to Windows, the X Window System on Unix and other environments. GLUT provides the utilities to create multiple windows, handle window system events like exposure after being obscured by another window, handle mouse and keyboard events, create text objects, and create pop-up menus, all through a system-independent API. GLUT is available for free download at http://reality.sgi.com/opengl/glut3/ as source code or precompiled for Windows. GLUT documentation and information are also available there.

The Fortran 90 bindings for OpenGL were approved by the OpenGL Architecture Review Board in 1998. These bindings together with the C API define the Fortran 90 API for OpenGL. The main features are:

- all program units that use an entity from OpenGL must USE module opengl_gl
- all procedure names are the same (except for being case insensitive)
- all procedure argument lists are the same
- all defined constants are the same
- variables and constants should specify the correct KIND to insure that Fortran and C types agree, for example real(GLFLOAT)

Thus the best way to learn how to use OpenGL from Fortran is to learn about OpenGL from a resource like the red book and apply the rules specified by the Fortran 90 bindings to determine the Fortran API. Studying the example programs that come with f90gl is also useful when getting started. f90gl is a public domain implementation of the Fortran 90 bindings. It provides an interface between a Fortran OpenGL program and the C libraries for OpenGL, GLU and GLUT. f90gl has been ported to most Unix platforms and most Fortran 90/95 compilers for Windows. The Fortran 90 bindings document and f90gl software are available from http://math.nist.gov/f90gl. f90gl comes as source code for all supported machines and precompiled for some Windows compilers.

Figure 2 illustrates a simple program using OpenGL and GLUT. This program draws a white square on a black background (see Figure 3) – it might be considered the "hello world" program of graphics.

The program begins by USEing modules opengl_gl and opengl_glut. It does not have to USE module opengl_glu because no facilities from GLU are used in this program. It then defines some constants for convenience later in the program. Note the use of the KIND specifiers glfloat and gldouble on both the type specification and the constant. Subroutine display is declared external here since it is an external subroutine and will be passed as an actual argument. One could just as well put display in a module and USE that module.

The first executable statement initializes GLUT. glutinit must be called exactly once in the program, and must be called before any other GLUT routine. Next the display mode is initialized. GLUT_SINGLE and GLUT_RGB are two of several symbolic constants that are or-ed together to determine the mode. Then a window is created. GLUT returns an integer identifier for the window, because it is possible to open multiple windows.

The next few statements initialize the view. First the background is set to black by setting the red, green and blue components of the color all to zero. The next three statements set the projection of three dimensional space onto the two dimensional window such that (0,1)X(0,1)X(-1,1) is mapped onto the window. By default the view is down the z axis. Other operations that would appear here in a more complicated program include setting lighting conditions and creating menus.

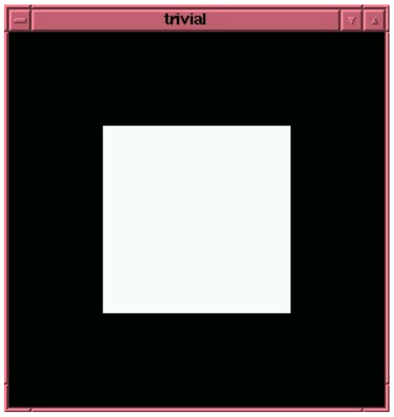
Then the callback functions are registered. These are subroutines that GLUT will call when some event occurs, like a mouse button is pressed or a menu item is selected. In this program the only callback function is the display function, which gets called whenever the window needs to be redisplayed (for example, after being un-iconified).

Finally, the glut main loop is entered. This subroutine never returns, allowing GLUT to process events until the program is terminated.

Subroutine display performs the actual drawing of the object. glclear sets the entire window to black, the color specified earlier by glclearcolor. The drawing color is then set to white by specifying red, green and blue to all be 1.0. Then the square is drawn. The call to glbegin says to draw a polygon consisting of all the vertices listed before the call to glend. Other values of the argument to glbegin allow a list of vertices to specify points, lines, triangles and quadrilaterals. Finally a call to glflush causes any OpenGL commands that have been buffered to execute immediately.

This simple example illustrates the basic structure of an OpenGL program

```
program trivial
use opengl_gl
use opengl_glut

real(kind=glfloat),  parameter :: zero  = 0.0_glfloat
real(kind=gldouble), parameter :: dzero = 0.0_gldouble, &
                                  one   = 1.0_gldouble
integer(kind=glcint) :: iwin
external display

! Initialize glut, set the display mode as single buffered and RGBA color,
! and open a window with "trivial" in its title bar.
call glutinit()
call glutinitdisplaymode(ior(GLUT_SINGLE,GLUT_RGB))
iwin = glutcreatewindow("trivial")

! select clearing (background) color to black
call glclearcolor(zero, zero, zero, zero)

! initialize viewing values
call glmatrixmode(GL_PROJECTION)
call glloadidentity()
call glortho(dzero, one, dzero, one, -one, one)

! Register callback function to display graphics.
call glutdisplayfunc(display)

! Enter main loop and process events.
call glutmainloop()

end program trivial

subroutine display()
use opengl_gl

! clear all pixels
call glclear(GL_COLOR_BUFFER_BIT)

! draw white polygon (rectangle) with corners at
! ((0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)
call glcolor3f(1.0_glfloat, 1.0_glfloat, 1.0_glfloat)
call glbegin(GL_POLYGON)
   call glvertex3f(0.25_glfloat, 0.25_glfloat, 0.0_glfloat)
   call glvertex3f(0.75_glfloat, 0.25_glfloat, 0.0_glfloat)
   call glvertex3f(0.75_glfloat, 0.75_glfloat, 0.0_glfloat)
   call glvertex3f(0.25_glfloat, 0.75_glfloat, 0.0_glfloat)
call glend()

! process buffered OpenGL routines
call glflush()

end subroutine display
```



and the general appearance of the API. Other examples that come with f90gl illustrate more advanced features like complex objects, lighting, animation, interaction and menus. The modview.f90 program contains a module that can be USEed by any program to add the interactive capabilities of rotate, pan and zoom with the mouse and arrow keys. With the standardization of the Fortran 90 bindings for OpenGL, the day of powerful and portable graphics from Fortran has arrived!