NISTIR 6134

A Fortran 90 Interface for OpenGL: Revised January 1998

William F. Mitchell
U. S. Department of Commerce
Technology Administration
National Institute of Standards and Technology
Information Technology Laboratory
Gaithersburg, MD 20899 USA

January 1998

A Fortran 90 Interface for OpenGL: Revised January 1998

William F. Mitchell*
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899
william.mitchell@nist.gov

Abstract

It is important to provide a good Fortran interface to OpenGL and related libraries for scientific visualization in mathematical software. OpenGL currently provides a Fortran interface which can be used by FORTRAN 77 or Fortran 90 programs. However, this interface relies upon several extensions to the FORTRAN 77 standard. By using the new features of Fortran 90 it is possible to define an interface to OpenGL that does not depend on any extensions to the standard and provides access to the full functionality of OpenGL. This document defines such an interface and supersedes NISTIR 5985.

1 Introduction

Most mathematical software for scientific computing is written in Fortran, and most scientific computing applications require 3D graphics for visualization. It is therefore important to provide a good Fortran interface to OpenGL and related libraries. OpenGL [5] currently provides a Fortran interface [1] which can be used by FORTRAN 77 or Fortran 90 programs. However, this interface relies upon several extensions to the FORTRAN 77 standard. Although some of these extensions are commonly used by Fortran compilers (e.g., REAL*4, REAL*8, INTEGER*4) and some have been made standard in Fortran 90 [2] (e.g., INCLUDE, identifiers up to 31 characters, underscore character in identifiers), others are not widely supported (e.g., LOGICAL*1, INTEGER*1, INTEGER*2, identifiers longer that 31 characters), which makes OpenGL difficult or impossible to use from some Fortran processors. Also, some of the OpenGL functionality cannot be achieved by any Fortran processor under the current Fortran binding (e.g., arbitrary length character string function result).

By using the new features of Fortran 90 it is possible to define an interface to OpenGL that does not depend on any extensions to the standard and provides access to the full functionality of OpenGL. It can also increase the capability of robustness and portability in the user application code, and increase the similarity between the Fortran and C interfaces.

*Contribution of NIST, not subject to copyright in the United States. OpenGL is a registered trademark of Silicon Graphics Computer Systems.

This document defines a Fortran 90 interface for OpenGL using features that are in Fortran 90 but not in FORTRAN 77. It is equally applicable to Fortran 95, and will most likely be valid for future Fortran standards. It is not intended to replace the existing Fortran interface (henceforth referred to as the FORTRAN 77 interface) at this time, since the existing interface will be required on systems that are still using a FORTRAN 77 compiler. The Fortran 90 interface is intended to provide an alternative through which the Fortran 90 programmer can achieve robustness and portability in an OpenGL application program. A reference implementation of the Fortran 90 interface has been made available to the public at http://math.nist.gov/f90gl.

This document supersedes NISTIR 5985 [4]. The Fortran 90 interface for OpenGL described in this document differs slightly from the one defined in NISTIR 5985. A summary of the changes is contained in Appendix A.

The major differences between the FORTRAN 77 and Fortran 90 interfaces are:

- The interface is accessed through modules, rather than INCLUDE lines. Among other advantages of modules, this provides explicit interfaces to the OpenGL procedures for improved robustness.

- Kind type parameters are provided for matching Fortran types to C types. This eliminates the need for nonstandard "*byte" declarations. It also provides a mechanism for transparently handling type mismatches on systems in which the Fortran processor does not support all the C types used by OpenGL, for increased portability.

- Fortran derived types are provided where C structs are used in the interface. This increases the similarity between the Fortran and C interfaces, and provides a mechanism through which the implementor can encapsulate whatever interface data is required.

- The Fortran functions corresponding to C functions that return a pointer to a character string now return a pointer to an array of characters. This increases the similarity between the Fortran and C interfaces, and adds the capability of arbitrary length character string return values.

- Extremely long names are truncated to 31 characters to comply with the Fortran 90 standard.

- The Fortran 90 interface to OpenGL is defined entirely on the Fortran side of the Fortran/C interface, so the issues associated with interoperability between Fortran and C are hidden from the user.

This interface explicitly covers the OpenGL 1.1 core library [5], and the GLU 1.2 library [6], but it also applies to earlier versions and it is anticipated that it will apply to later versions. The principles laid out in this interface can also be applied to related libraries, toolkits, and OpenGL extensions. Some entities from the Graphics Library Utility Toolkit (GLUT) Version 3 [3] and the OpenGL tk toolkit are used for illustration in this document, even though they are not part of OpenGL proper.

2 Interface Definition

This section describes and discusses the Fortran 90 interface to OpenGL.

2.1 Modules

The Fortran 90 interface to OpenGL is accessed through modules. The modules provide access to kind type parameters, named constants, procedures, and derived types (structures).

The module OPENGL_KINDS contains the definitions of the kind type parameters as described in Section 2.2. This module is not normally used directly in application code, but is inherited through the other modules. The kind type parameters are defined as integers of default kind with the PARAMETER attribute.

The module OPENGL_GL provides access to the core OpenGL library procedures and named constants, and the definitions in OPENGL_KINDS. It may also provide access to one or more OpenGL extensions, along with the related named constants and derived types.

Additional modules provide access to related libraries, and are given a descriptive name beginning with OPENGL. For example, the module OPENGL_GLU contains the procedures, named constants and derived types for the OpenGL Utility Library (GLU).

2.2 Types

2.2.1 Numeric

The correspondence between Fortran and C numeric types is achieved through use of kind type parameters. The module OPENGL_KINDS contains the definition of these parameters such that the C representation of an entity of a given OpenGL type agrees with the Fortran representation of an entity of the corresponding type and kind whenever possible. When the corresponding representation is not provided by the Fortran processor, the lack of said representation remains transparent to the user.

The OpenGL numeric types and the corresponding Fortran 90 TYPE(KIND) are:

GLbyte	INTEGER(GLBYTE)
GLubyte	INTEGER(GLUBYTE)
GLshort	INTEGER(GLSHORT)
GLushort	INTEGER(GLUSHORT)
GLint	INTEGER(GLINT)
GLuint	INTEGER(GLUINT)
GLenum	INTEGER(GLENUM)
GLbitfield	INTEGER(GLBITFIELD)
GLsizei	INTEGER(GLSIZEI)

GLfloat	REAL(GLFLOAT)
GLclampf	REAL(GLCLAMPF)
GLdouble	REAL(GLDOUBLE)
GLclampd	REAL(GLCLAMPD)

The user's code should always specify the kind type parameter for all actual arguments passed to OpenGL procedures to ensure correspondence between C and Fortran types and portability of the user's code:

- Variables should have the kind type parameter in the declaration
- Constants should have the kind type parameter attached (e.g., 1.0_GLFLOAT)
- Expressions should evaluate to a value with the appropriate kind type parameter

The Fortran standard does not specify what kind type parameters are to be provided for each type. It is possible that some OpenGL types do not have a corresponding TYPE(KIND) on a given Fortran processor. On current systems this is highly unlikely for the float, double and long integer types, but may occur with the short integer types. In this case, the implementation of the interface will match Fortran and C types in a manner that is transparent to the user. There are at least two approaches that can be taken for this. In the first approach the interface accesses the OpenGL library routine that accepts the available type, rather than the type expected according to the procedure name. In the second approach the C procedure that is called by the Fortran procedure converts the arguments to the type specified by the OpenGL definition. If there are any return values of the missing type, they are converted to the available type before returning to the Fortran procedure.

For example, suppose GLshort is a 2-byte integer, GLint is a 4-byte integer, and the Fortran compiler supports 4-byte integers but not 2-byte integers, and assume the Fortran 90 interface is implemented by a set of "wrapper" functions. Then GLSHORT will be set to the same value as GLINT, which is the kind type parameter such that INTEGER(GLINT) is a 4-byte integer. Consider an invocation of glVertex2s. In the first approach, the wrapper function simply invokes the C glVertex2i. In the second approach, the C procedure invoked by the Fortran procedure will accept an argument of type GLint, convert it to type GLshort, and invoke the C glVertex2s.

For the user's application code, this is all transparent. The user declares the argument to be of type INTEGER(GLSHORT). If the equivalent of a GLshort is supported by the Fortran processor, then the short integer is used; if not, then the equivalent of GLint is used with one of the above methods for handling mismatched type. The user's code works in both environments unchanged.

Note that the equivalent of GLbyte (probably a 1-byte integer) may be supported by the Fortran processor, may require promotion to the kind GLSHORT, or may require promotion to the kind GLINT depending on what kinds of integers are supported by the Fortran processor.

2.2.2 Logical

The OpenGL logical type and the corresponding Fortran 90 TYPE(KIND) is:

GLboolean   LOGICAL(GLBOOLEAN)

The type GLboolean is typically a 1-byte entity with the value 0 representing false and nonzero representing true. The Fortran processor may or may not support a 1-byte logical type. The kind type parameter GLBOOLEAN, defined in the module OPENGL_KINDS, is normally set to the kind type parameter for a 1-byte logical if it is supported, or the default kind type parameter for LOGICAL if it is not. If the 1-byte logical is not supported, or the Fortran representation of logical values does not correspond to the C representation, then the interface routines will perform appropriate type conversions similar to the type conversions described in the section on numeric types.

2.2.3 Character

Some procedures in related libraries and toolkits have character string arguments. In the Fortran 90 interface, the dummy argument is given the type CHARACTER(LEN=*). At the Fortran 90 interface, this is a Fortran character string. Any conversion required to match the C character string (such as null termination) is handled by the implementation of the interface.

OpenGL functions that return a character string are also no problem in Fortran 90. In C the resulting string can be arbitrarily long. In Fortran, this is obtained by declaring the function result to be a pointer to an array of type CHARACTER(LEN=1), and allocating the pointer inside the function. The user can obtain the number of characters using the SIZE intrinsic function, and, if the result is assigned to a pointer variable, can deallocate the memory.

2.2.4 Pointer

Some OpenGL procedures, or procedures in related libraries and toolkits, may require the user to maintain the value of a C pointer. Fortran does not provide pointers in this sense, so this use of pointers is restricted to obtaining a C pointer from an OpenGL procedure, and passing it to another procedure as an actual argument. Thus what is required is a means of storing the bit patterns contained in C pointer variables. The user may also copy a C pointer from one variable to another, which precludes the use of numeric types which are allowed to change the representation (for example, by normalizing the exponent). In the Fortran 90 interface, the derived type TYPE(GLCPTR) is provided for storing C pointers. An implementation of the interface is free to define the components of this type in any way that meets the requirements.

Some applications require that a C pointer be compared to NULL, thus a null pointer value of TYPE(GLCPTR) is provided, and the operator == is extended to compare two variables of TYPE(GLCPTR). The null pointer value is a named constant defined in module OPENGL_KINDS, is called GLNULLPTR, and has a value that is equal to the C NULL pointer in the sense that the operator == will return .TRUE. if GLNULLPTR is compared to a variable of TYPE(GLCPTR) that has been assigned the C NULL pointer.

2.2.5 Structures

Some related libraries and toolkits define structures that are used as arguments to the procedures. The Fortran 90 interface defines derived types corresponding to these structures. The name of the derived type is obtained from the name of the structure, subject to the same name modification rules used for the Fortran 90 procedure names in section 2.3. The derived type definitions are contained in the module for the given library or toolkit. The components of the derived type contain whatever information is required to fulfill the specification of the procedures that operate on that type. Since some components may be useful to the user, they have the PUBLIC attribute. An example of where the components are useful to the user is provided by the tk toolkit where a tk procedure sets the components of a derived type, and a GLU procedure needs the values in the components:

```
TYPE(TK_RGBImageRec), POINTER ::  IMAGE
IMAGE => tkRGBImageLoad( TABLE_TEXTURE )
ERR = gluBuild2DMipmaps(GL_TEXTURE_2D, 3_GLINT, IMAGE%sizeX, &
    IMAGE%sizeY, GL_RGB, GL_UNSIGNED_BYTE, IMAGE%data)
```

For functions that return a C pointer to an OpenGL struct, the Fortran 90 function returns a Fortran pointer of the derived type. (This is an exception to the rule for handling pointers given in section 2.2.4.) If the C pointer is NULL, then the Fortran pointer is nullified (disassociated), so that the C test "if (cptr == NULL)" is achieved with "IF (.NOT. ASSOCIATED(fptr))", where cptr and fptr are pointer variables in C and Fortran, respectively.

For example, consider the GLU type gluQuadricObj. The Fortran 90 type

```
TYPE gluQuadricObj
  TYPE(GLCPTR) ::  addr
  !  there may be other components
END TYPE gluQuadricObj
```

is defined in module OPENGL_GLU. The function gluNewQuadric would have the effect of

```
FUNCTION gluNewQuadric()
TYPE(gluQuadricObj), POINTER ::  gluNewQuadric
ALLOCATE(gluNewQuadric)
gluNewQuadric%addr = c_gluNewQuadric()
IF (gluNewQuadric%addr == GLNULLPTR) THEN
  DEALLOCATE(gluNewQuadric)
  NULLIFY(gluNewQuadric)
ENDIF
END FUNCTION gluNewQuadric
```

2.2.6 Void

Many OpenGL procedures use the type GLvoid for an argument that may be one of several different types. Generic interfaces provide this capability in Fortran 90. Procedures with a GLvoid argument have a generic interface (with the usual name for the procedure as defined in section 2.3) to a set of specific routines, one for each type specified by the OpenGL definition. Additionally, it interfaces to a specific routine that accepts an argument of type TYPE(GLCPTR) to allow the GLvoid argument to be a C pointer returned by a prior call to an OpenGL procedure.

Processors that do not support the short integers require additional work here, but it remains transparent to the user. Consider the situation where the Fortran processor does not support the kind of integer that corresponds to GLshort. Then GLSHORT is the same as GLINT, so there is no specific routine for the type INTEGER(GLSHORT). If the user passes an argument of type INTEGER(GLSHORT), then the specific routine that is called is the one with dummy argument of type INTEGER(GLINT). But, in all such core OpenGL and GLU routines there is another argument that tells what type the GLvoid argument is to be interpreted as. If that argument indicates that the user is passing a INTEGER(GLSHORT), but the specific routine for INTEGER(GLINT) is called because GLSHORT is the same as GLINT, then the interface will handle the mismatched types as described in section 2.2.1. The situation is similar for GLBYTE, except that GLBYTE could be either GLSHORT or GLINT, depending on the Fortran processor.

2.3 Procedures

All OpenGL procedures are available in the Fortran 90 interface. The argument lists and return values are identical, subject to the equivalences described in section 2.2. C functions of type void are Fortran subroutines; C functions of other types are Fortran functions of the corresponding type. All of the procedure names corresponding to OpenGL procedures are generic names in the Fortran 90 interface.

The procedure names in the Fortran 90 interface are derived from the C names as follows:

- Case is insignificant. This conforms to the Fortran 90 requirement that lower case letters are equivalent to the corresponding upper case letters except in a character context.

- Any names that are longer than 31 characters are truncated to 31 characters. This conforms to the Fortran 90 requirement that the maximum length of a name is 31 characters. There are no names that require truncating in the core OpenGL and GLU libraries.

2.4 Defined constants

All OpenGL defined constants are provided as named constants in module OPENGL_GL. These are integers of the appropriate kind with the PARAMETER attribute and the same value as in the C interface.

The names for the Fortran 90 named constants are derived from the OpenGL defined constants as follows:

- Case is insignificant.
- Any names that are longer than 31 characters are truncated to 31 characters. There are no names that require truncating in the core OpenGL and GLU libraries.
- Any names that are not unique after discarding case are replaced with a suitable descriptive name. Specifically, the tk toolkit contains lower case key constants, TK_a through TK_z, and upper case key constants, TK_A through TK_Z. In module OPENGL_TK the lower case key constants are named TK_LC_A through TK_LC_Z, with LC standing for lower case. There are no case dependent defined constants in the core OpenGL and GLU libraries.

2.5 Dummy procedures

Some routines in related libraries and toolkits take a procedure as an argument. Whenever possible the explicit interface of these routines provided by the Fortran 90 interface to OpenGL provides an explicit interface for the dummy procedure. When this is not possible (for example, when the dummy procedure contains an argument of C type (void *) for which there is more than one valid type), the dummy procedure has an implicit interface. There are no routines in the core OpenGL, GLU and GLUT libraries that require a dummy procedure to have an implicit interface.

When the argument is used as a callback function, the procedure may allow NULL as the value of the argument to indicate that the corresponding callback is to be disabled. For example, GLUT uses this technique. When this is the case, the Fortran 90 interface for this library provides the symbol *library-prefix*NULLFUNC which can be passed in place of NULL. For example, the interface to GLUT provides the symbol GLUTNULLFUNC. Each library requires its own NULLFUNC in order to preserve the independence of the modules corresponding to each library.

2.6 Array arguments

The explicit interfaces of the Fortran 90 interface declare array arguments to be assumed-size arrays, i.e., declared with DIMENSION(*). They are not assumed-shape arrays, declared with DIMENSION(:), because most Fortran 90 processors pass assumed-shape arrays as dope vectors containing the dimensions of the array in addition to the starting address. The wrappers would thus be more complicated, to extract the address from the dope vector, and less portable since there is no standard for the dope vectors. There is no loss of functionality by using assumed-size arrays.

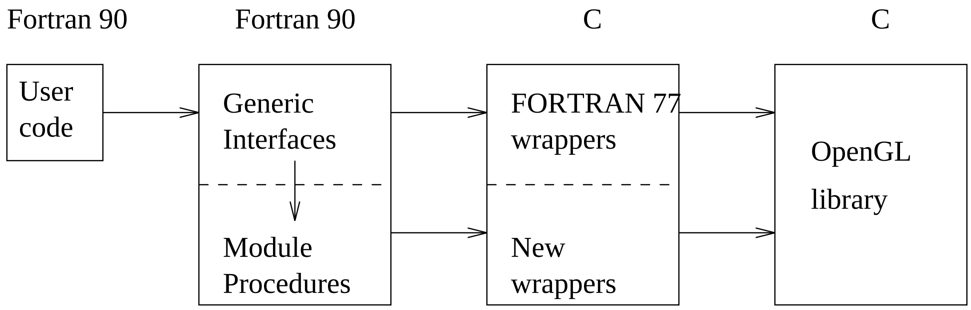

Figure 1: Example implementation using wrappers.

3 Implementation

In the FORTRAN 77 binding, the user calls C functions from the Fortran program, leading to portability issues and the requirement for the binding to address the interfacing of Fortran and C procedures. The Fortran 90 interface to OpenGL does not address this issue. The user interface is entirely on the Fortran side of the Fortran/C interface, therefore the Fortran/C interface is contained entirely inside the Fortran 90 interface to OpenGL. It is anticipated that most vendor implementations will be for a specific system with specific Fortran and C compilers. The containment of the Fortran/C interface leaves these implementors free to use whatever system dependent techniques are required for the Fortran/C interface without affecting the interface to the user application code. In the case of an implementor attempting to provide an implementation that is portable over several Fortran/C/OS combinations, it is left to the implementor to determine how to achieve portability, however the reference implementation may be a useful guideline.

There is no requirement on the actual architectural design of the Fortran 90 interface to OpenGL. The only requirement is that the aforementioned modules be provided, and that they provide access to the kind type parameters, procedures, named constants, and derived types described above. However it is anticipated that most implementations will simply provide "wrapper" functions on top of an existing OpenGL implementation. Here the wrapper functions would most likely fall on both the Fortran and C sides of the interface. An example of how this might be implemented is illustrated in figure 1.

In this approach, the Fortran 90 names for all the OpenGL procedures are defined in generic interfaces in module OPENGL_GL. Some of them are used simply to rename the existing FORTRAN 77 interface. Other generic interfaces may include interfaces to module procedures which call new wrapper functions. In particular, this would be used when type conversions are used because the Fortran processor does not support the requested type or kind, when one of the arguments is of type GLvoid with several valid types for that argument, or when one of the arguments is a derived type.

In this example, module OPENGL_GL would also contain the definition of all the named constants as integer values with the PARAMETER attribute and would also use module

OPENGL_KINDS and provide access to the definitions in OPENGL_KINDS to any program unit that uses OPENGL_GL.

4 Potential problems

4.1 Assumptions on compilers

The Fortran 90 interface to OpenGL is considerably more robust and portable than the FORTRAN 77 interface. However, until there is a standard for inter-language calling sequences, it must be assumed that the compilers provide a sufficient inter-language calling convention. Most Fortran 90 and C compilers satisfy this requirement in a processor dependent manner. The next revision of the Fortran standard is planned to incorporate interoperability features to facilitate calling C procedures from Fortran programs. These interoperability features should allow a portable implementation of the Fortran 90 interface for OpenGL.

4.2 Preserved addresses

Some OpenGL and GLU procedures internally set a C pointer to one of the arguments so that the argument can be used by a different procedure called later. In the Fortran 90 interface, these arguments are assumed-size arrays with the TARGET attribute. It is the user's responsibility to insure that the actual argument will persist and may be modified by a procedure to which it is not an argument. One means of achieving this is to pass a whole array with the TARGET and SAVE attributes as the actual argument. Note that using an expression or array section in this context is likely to fail. The OpenGL 1.1 core and GLU procedures and arguments effected by this are:

glFeedbackBuffer	buffer
glSelectBuffer	buffer
glEdgeFlagPointer	pointer
glTexCoordPointer	pointer
glColorPointer	pointer
glIndexPointer	pointer
glNormalPointer	pointer
glVertexPointer	pointer
gluNurbsCurve	uknot, ctlarray
gluNurbsSurface	uknot, vknot, ctlarray
gluPwlCurve	array
gluTessVertex	data

4.3 Unsigned int

Fortran does not provide unsigned integer types; signed integers of the same size are used for these types. The Fortran intrinsic function IBSET can be used for setting values in which the leading bit is a 1. For example, the hexadecimal pattern 8000000A can be set in either an assignment statement or an initialization expression using IBSET as follows:

```
INTEGER(GLUINT) ::  U = IBSET(10,31) !  use bit pattern for 10 and set 31st bit
```

4.4 Array order

The user should remember that Fortran stores multidimensional arrays in column major order, whereas C stores them in row major order. Some multidimensional Fortran arrays may require transposition. The exception is the transformation matrices passed to glLoadMatrix and glMultMatrix which, as a 4x4 array, are assumed to be in column major order.

5 System installation

The location of the software for the Fortran 90 interface to OpenGL is system dependent. The OpenGL documentation provides this information for the user.

5.1 Libraries

The Fortran 90 interface procedures may be placed in either the same libraries as the OpenGL procedures (e.g., libGL, libGLU, etc.) or in separate libraries (e.g., libf90GL, libf90GLU, etc.).

5.2 Module files

Many Fortran 90 compilers generate a file containing module information. The name of the file is usually the module name followed by a compiler dependent suffix, for example opengl_gl.mod. If the compiler generates module files, these are located in the same directory as the OpenGL include files (e.g., gl.h). Some Fortran 90 compilers provide a command line option for specifying the location of module files (e.g., -I); with other compilers the module files will have to be copied (or linked) to the user's source code directory.

6 Reference implementation

A reference implementation of the Fortran 90 interface for OpenGL is available in the software package called f90gl available from http://math.nist.gov/f90gl. Version 1.1 of the reference implementation covers the OpenGL 1.1 core, GLU 1.2, GLUT 3.6, and some extensions.

References

[1] Allen Akin, *OpenGL FORTRAN Binding Proposal*, http://www.sgi.com/Technology/openGL/fortran.html

[2] ANSI, *American National Standard for Programming Language – Fortran – Extended*, ANSI, New York, 1992.

[3] Mark J. Kilgard, *The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3*, http://reality.sgi.com/mjk_asd/spec3/spec3.html

[4] William F. Mitchell, *A Fortran 90 Interface for OpenGL*, NISTIR 5985, 1997.

[5] Mark Segal and Kurt Akeley, *The OpenGL Graphics System: A Specification (Version 1.1)*, http://www.sgi.com/Technology/openGL/glspec1.1/glspec.html

[6] Kevin P. Smith and Chris Frazier, *The OpenGL Graphics System Utility Library*, http://reality.sgi.com/mjk_asd/GLUspec.ps.gz

A Differences from NISTIR 5985

This document supersedes NISTIR 5985 [4]. The Fortran 90 interface for OpenGL described in this document differs slightly from the one defined in NISTIR 5985. This appendix contains a summary of the changes.

- Removed f90 prefix from procedure names, kind type parameters, and derived types. Fortran processors are required to keep module procedure names distinct from other instances of the same name, so the f90 prefix is not required to avoid name space clashes. Similarly, there can be no conflicts with kind type parameters and derived types in the modules.
- Renamed modules to start with OPENGL_ instead of f90gl. With the f90 prefix removed from other entities, the module names were the only names beginning with f90. It was suggested that f90 be removed from the names to avoid possible confusion about the applicability of this interface to Fortran 95 and future versions of Fortran.
- Explicitly state that the module OPENGL_GL provides access to the definitions in the module OPENGL_KINDS. This was ambiguous.
- Changed the definition of GLCPTR (a type for storing C pointers) to a derived type, relaxed the requirements on the definition of the null pointer, and extended == to compare TYPE(GLCPTR). The previous requirement using character strings to store C pointers can be inefficient on some processors due to alignment requirements. Hiding it inside a derived type gives the implementor more flexibility. The explicit definition of the value of the null pointer may not agree with some processors.
- Replaced the mixed PUBLIC/PRIVATE recommendation for components of derived types with the requirement that the components be PUBLIC. Mixed PUBLIC/PRIVATE attributes in a derived type are not standard conforming.
- Added the requirement that the OpenGL procedure names be generic names in the interface. The distinction between generic and specific names is great enough that it should be specified if the names are to be generic. Some of the procedures require a generic interface, so for consistency they are all to be generic.
- Changed the interface of dummy procedures from implicit to explicit. Explicit interfaces are preferable whenever possible, and currently there are no examples of dummy procedures that require an implicit interface.
- Relaxed the definition of NULLFUNC to be a symbol that is not necessarily an external procedure. This removes a potential conflict between NULLFUNC and the explicit interface of the associated dummy procedure.
- Modified the discussion of OpenGL procedures saving pointers to dummy arguments (section 4.2), and added the requirement that such arguments have the TARGET attribute.
- Removed suggested use of BOZ to set unsigned integers. Setting the highest order bit with BOZ is not standard conforming.

- Editorial changes that do not affect the definition of the interface.