The OpenGL™ Character Renderer: A Specification

Craig Dunwoody
Silicon Graphics

Version 0.2 – 7 Mar 1996

Contents

1 Introduction

This document defines the OpenGL™ Character Renderer (GLC), a state machine that provides OpenGL[1] programs with character rendering services via an application programming interface (API). This document is written with the assumption that the reader understands the OpenGL specification.

This section provides an overview of GLC, examples of its use, and a comparison with other character rendering services. Section 2 defines the GLC machine, Section 3 defines the GLC API, and Section 4 covers future extensions to GLC.

1.1 Overview

A GLC *client* is a program that uses OpenGL (henceforth, "GL") and GLC. A GLC *library* is a subroutine library that implements GLC. Figure 1 shows that like the GL Utilities (GLU), GLC is implemented on the

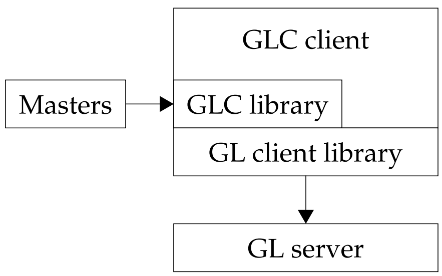

Figure 1: Relationship between GLC and GL

client side of the GL client-server connection and performs all of its rendering by issuing GL commands.

A GLC *context* is an instantiation of GLC. When a client thread issues a GLC command, the thread's *current* GLC context executes the command.

A client can render a character by issuing the command **glcRenderChar (GLint inCode)**. GLC finds a *font* that *map*s **inCode** to a character such as LATIN CAPITAL LETTER A, then uses one or more *glyph*s from the font to create a graphical *layout* that represents the character. Finally, GLC issues a sequence of GL commands to draw the layout. Glyph coordinates are defined in *em units* and are transformed during rendering to produce the desired mapping of the glyph shape into the GL window coordinate system.

In addition to commands for rendering, the GLC API includes measurement commands that return certain *metrics* (e.g., bounding box) for a layout. Since the focus of GLC is on *rendering* and not *modeling*, the GLC API does not provide access to glyph shape data.

A font is a stylistically consistent set of glyphs that can be used to render some set of characters. Each font has a *family* name (e.g. Palatino) and a state variable that selects one of the *face*s (e.g. Regular, Italic, Bold, Bold Italic) that the font contains. A *typeface* is the combination of a family and a face (e.g. Palatino Bold).

A font is an instantiation of a *master*, which is a representation of the font that is stored outside of GLC in a standard format such as TrueType[2] or Type1[3]. A *catalog* is a named list of masters, which may be implemented as a file system directory containing master files. A list of catalog names defines the list of masters that can be instantiated in a GLC context.

Before issuing a GLC rendering command a client must issue GL commands directly to establish a GL state such that the GL commands issued by GLC produce the desired result. For example, before issuing a **glcRenderChar** command a client may issue **glColor** and **glRasterPos** commands to establish the color and origin of the resulting layout.

1.2 Examples

At this point some examples of the use of GLC may be helpful to the reader, even though the GLC commands used therein have not yet been defined. The following ISO C code fragment uses GL and GLC to render the character LATIN CAPITAL LETTER A in red at (100, 100) using a default typeface at a scale of 24 pixels per em. In this example, GLC issues a **glBitmap** command to draw the layout.

```
glcContext ( glcGenContext () );
glcScale ( 24.f, 24.f );
glColor3f ( 1.f, 0.f, 0.f );
glRasterPos2f ( 100.f, 100.f );
glcRenderChar ( 'A' );
```

In the following example, GLC renders the string “Hello, world!” in red 24 pixel Palatino Bold at a rotation of 30 degrees, starting at (100, 100).

```
glcContext ( glcGenContext () );
glcFont ( glcNewFontFromFamily ( 1, "palatino" ) );
glcFontFace ( 1, "bold" );
glcScale ( 24.f, 24.f );
glcRotate ( 30.f );
glColor3f ( 1.f, 0.f, 0.f );
glRasterPos2f ( 100.f, 100.f );
glcRenderString ( "Hello, world!" );
```

1.3 GLC vs. Other Services

A GL implementation may support a platform specific interface (PSI) such as GLX or WGL. A PSI may provide character rendering services. For example, the command **glXUseXFont** extracts glyphs from an X Window System font object, creating for each glyph a GL list object containing a **glBitmap** command. A client can then issue the command **glCallLists** to render characters using the extracted glyphs.

The character rendering services provided by GLC are superior in many ways to those provided by PSIs. The most significant advantages of GLC over PSI character rendering services are as follows:

1. The GLC API is platform independent. Since most nontrivial GL applications render characters, GLC is an important step toward the goal of truly portable GL applications.

2. The GLC reference implementation is platform independent. GLC implementations and future extensions can be made available on all GL platforms with little effort.

3. GLC is simpler to use. Only two lines of GLC commands are required to prepare for rendering characters.

4. GLC provides more ways to exploit the rendering power of GL. For example, a glyph can be drawn as a bitmap, a set of lines, a set of triangles, or a textured rectangle.

5. GLC provides better support for glyph transformations. For example, GLC supports rotated text, which is unavailable in GLX.

6. GLC provides better support for the large coded character set defined by the standards ISO/IEC 10646-1:1993[4] and Unicode 1.1[5,6,7].

7. GLC commands can be represented in GLS[8] streams. This will enable resolution independent cap-

ture, interchange, and printing of GL pictures containing text.

2 Machine Definition

2.1 Data Types

GLC uses GL data types and defines additional types, which are listed in Table 1.

Type	Description
GLCchar	Character code array element
GLCenum	32 bit enumerant
GLCfunc	Callback function pointer

Table 1: GLC data types

2.2 Errors

Each client thread has a private GLC error code variable of type **GLCenum**. The initial value of this variable is **GLC_NONE**. If a GLC command raises an error, and the value of this variable is **GLC_NONE**, the command stores the error's code in the variable.

If a GLC command raises an error, the command will have no effect except the possible storing of the error's code. GLC error semantics apply only to GLC errors and not to GL errors or system errors (e.g., memory access errors) that occur during GLC command execution.

Every GLC command execution begins with a test to determine if the command parameters are valid. If this test fails, the command raises **GLC_PARAMETER_ERROR**. Otherwise, GLC performs a test to determine if its state is such that the command is valid. If this test fails, the command raises **GLC_STATE_ERROR**. Otherwise, command execution proceeds. If at any point during command execution a needed resource (e.g., memory) is unavailable, the command raises **GLC_RESOURCE_ERROR**.

2.3 Contexts

Each GLC context has a nonzero ID of type **GLint**. When a client is linked with a GLC library, the library maintains a list of IDs that contains one entry for each of the client's GLC contexts. This list is initially empty.

Each client thread has a private GLC context ID variable that always contains either the value zero, indicating that the thread has no current GLC context, or the ID of the thread's current GLC context. The initial value of this variable is zero.

When the ID of a GLC context is stored in the GLC context ID variable of a client thread, the context is said to be current to the thread. It is not possible for a GLC context to be current simultaneously to multiple threads.

With the exception of the per-thread GLC error code and context ID variables, all of the GLC state variables that are used during the execution of a GLC command are stored in the issuing thread's current GLC context.

Each GLC command belongs to one of the following categories: Global, Context, Master, Font, Transformation, Rendering, and Measurement. Global commands do not use GLC context state variables and can therefore be executed successfully if the issuing thread has no current GLC context. All other GLC commands raise **GLC_STATE_ERROR** if the issuing thread has no current GLC context.

This document identifies explicitly the situations in which GLC may issue GL commands. In some GL implementations, the execution behavior of a GL command is defined only if the GL client has previously created a GL context and made it current to the issuing thread. It is the responsibility of the GLC client to set up the underlying GL implementation such that whenever GLC issues a GL command, the execution behavior of that command is defined.

The behavior of GLC depends on the extension set and version of the underlying GL implementation. When a GLC context is made current to a thread, GLC issues the commands

glGetString (GL_VERSION);
glGetString (GL_EXTENSIONS);

and stores the returned strings.

2.4 Character codes

Except where otherwise specified, every character code used in GLC is an element of the Universal Multiple-Octet Coded Character Set (UCS) defined by the standards ISO/IEC 10646-1:1993 and Unicode 1.1. A UCS code is denoted as U+*hexcode*, where *hexcode* is a sequence of hexadecimal digits. Each UCS code corresponds to a character that has a unique name string. For example, the code U+41 corresponds to the character LATIN CAPITAL LETTER A.

2.5 Constants

GLC defines a set of implementation specific constants. The integer constants **GLC_VERSION_MAJOR** and **GLC_VERSION_MINOR** identify the version of GLC that the implementation supports. These constants correspond to a version *A*.*B* of this document. If a new version breaks compatibility, the major version number *A* will be incremented by one. Otherwise, the minor version number *B* will be incremented by one.

The string constant **GLC_VENDOR** identifies the vendor of the implementation. If the vendor offers a GL implementation, the value of **GLC_VENDOR** must equal the value of the GL implementation constant **GL_VENDOR**. Otherwise, the value of **GLC_VENDOR** must be a unique name that the vendor has allocated in the GLC vendor registry by sending email to glc-registry@sgi.com.

The string constant **GLC_EXTENSIONS** lists in alphabetical order the names of the GLC extensions that are supported by the implementation. One SPACE (U+20) separates each pair of adjacent names. Example: “GLC_EXT_kern GLC_SGI_ligature”.

Each extension name begins with “GLC_*prefix*_”. Iff two or more vendors support the extension, the value of *prefix* is “EXT”. Otherwise, the value of *prefix* is a vendor specific extension name prefix string, optionally followed by one ‘X’ (U+58). The ‘X’ indicates that the extension is experimental and may be redefined or removed in future releases. One ‘_’ (U+5F) should be used between each pair of adjacent words in an extension name.

If a vendor offers a GL implementation, the vendor’s GLC extension name prefix must be the same as the vendor’s GL extension name prefix. Otherwise, this prefix must be a unique string that the vendor has allocated in the GLC extension name prefix registry by sending email to glc-registry@sgi.com. It is common for the GLC extension name prefix to equal the value of **GLC_VENDOR**.

The string constant **GLC_RELEASE** identifies the vendor specific software release that contains the implementation. Example: “IRIX 6.2”.

2.6 Masters

Section under construction.

2.7 Fonts

Section under construction.

2.8 Rendering

Section under construction.

2.9 Measurement

Section under construction.

3 API Definition

3.1 Errors

If an a GLC command raises an error, and the command returns a value, the returned value will be zero.

3.2 Character codes

In the GLC API, a single character code is represented as a value of type **GLint**.

3.3 Strings

Except where otherwise specified, every character string used in the GLC API is represented as a zero terminated array of UCS code values. The type of these values is declared statically as **GLCchar**, but the actual type of these values is determined dynamically by the value of the variable **GLC_STRING_TYPE**. The values **GLC_UCS1**, **GLC_UCS2**, and **GLC_UCS4** specify that each value is of type **GLubyte**, **GLushort**, or **GLint**, respectively. The initial value of **GLC_STRING_TYPE** is **GLC_UCS1**.

Some GLC commands return strings. The return value of these commands is a pointer to a string return buffer in the issuing thread's current GLC context. This pointer is valid until the next GLC command is issued. The pointer may be used as a parameter to the next GLC command. The client must copy the returned string to a client provided buffer if the value of the string will be needed after the next GLC command is issued.

The value of a character code in a returned string may exceed the range of the character encoding selected by **GLC_STRING_TYPE**. In this case, the returned character is converted to a character sequence **<***hexcode***>**, where \ is the character REVERSE SOLIDUS (U+5C), **<** is the character LESS-THAN SIGN (U+3C), **>** is the character GREATER-THAN SIGN (U+3E), and *hexcode* is the original character code represented as a sequence of hexadecimal digits. The sequence has no leading zeros, and alphabetic digits are in upper case.

3.4 Global commands

The commands described in this section do not use GLC context state variables and can therefore be executed successfully if the issuing thread has no current GLC context. All other GLC commands raise **GLC_STATE_ERROR** if the issuing thread has no current GLC context.

3.4.1 glcContext

The command

void glcContext (GLint inContext)

assigns the value **inContext** to the issuing thread's current GLC context ID variable. The command raises **GLC_PARAMETER_ERROR** if **inContext** is nonzero and is not the ID of one of the client's GLC contexts. The command raises **GLC_STATE_ERROR** if **inContext** is the ID of a GLC context that is current to a thread other than the issuing thread. The command raises **GLC_STATE_ERROR** if the issuing thread is executing a callback function that has been called from GLC.

3.4.2 glcDeleteContext

The command

void glcDeleteContext (GLint inContext)

marks for deletion the GLC context identified by **inContext**. If the marked context is not current to any client thread, the command deletes the marked context immediately. Otherwise, the marked context will be deleted during the execution of the next **glcContext** command that causes it to not be current to any client thread. The command raises **GLC_PARAMETER_ERROR** if **inContext** is not the ID of one of the client's GLC contexts.

3.4.3 glcGenContext

The command

GLint glcGenContext (void)

generates a new GLC context and returns its ID.

3.4.4 glcGetAllContexts

The command

GLint* glcGetAllContexts (void)

returns a zero terminated array of GLC context IDs that contains one entry for each of the client's GLC contexts. GLC uses the ISO C library command **malloc** to allocate the array. The client should use the ISO C library command **free** to deallocate the array when it is no longer needed.

3.4.5 glcGetCurrentContext

The command

GLint glcGetCurrentContext (void)

returns the value of the issuing thread's current GLC context ID variable.

3.4.6 glcGetError

The command

GLCenum glcGetError (void)

retrieves the value of the issuing thread's GLC error code variable, assigns the value **GLC_NONE** to that variable, and returns the retrieved value. Table 1 lists the GLC error codes.

Name	Enumerant
GLC_NONE	0x0000
GLC_PARAMETER_ERROR	0x0040
GLC_RESOURCE_ERROR	0x0041
GLC_STATE_ERROR	0x0042

Table 2: Error codes

3.4.7 glcIsContext

The command

GLboolean glcIsContext (GLint inContext)

returns **GL_TRUE** iff **inContext** is the ID of one of the client's GLC contexts.

3.5 Context commands

3.5.1 glcCallbackFunc

The command

void glcCallbackFunc (GLCenum inOpcode, GLCfunc inFunc)

assigns the value **inFunc** to the callback function variable identified by **inOpcode**. Table 1 lists the GLC call-

Name	Enumerant	Initial value	Type signature
GLC_OP_glcUnmappedCode	0x0020	**GLC_NONE**	**GLboolean (*)(GLint inCode)**

Table 3: Callback function variables

back function variables.

3.5.2 glcDataPointer

The command

void glcDataPointer (GLvoid* inPointer)

assigns the value **inPointer** to the variable **GLC_DATA_POINTER**.

3.5.3 glcDeleteGLObjects

The command

void glcDeleteGLObjects (void)

causes GLC to issue a sequence of GL commands to delete all of the GL objects that it owns. GLC uses the command **glDeleteLists** to delete all of the GL objects named in **GLC_LIST_OBJECT_LIST** and uses the command **glDeleteTexturesEXT** to delete all of the GL objects named in **GLC_TEXTURE_OBJECT_LIST**. When an execution of **glcDeleteGLObjects** finishes, both of these lists will be empty.

3.5.4 glcDisable

The command

void glcDisable (GLCenum inAttrib)

assigns the value **GL_FALSE** to the boolean variable identified by **inAttrib**. Table 1 lists the GLC boolean

Name	Enumerant	Initial value
GLC_AUTO_FONT	0x0010	**GL_TRUE**
GLC_GL_OBJECTS	0x0011	**GL_TRUE**
GLC_MIPMAP	0x0012	**GL_TRUE**

Table 4: Boolean variables

variables.

3.5.5 glcEnable

The command

void glcEnable (GLCenum inAttrib)

assigns the value **GL_TRUE** to the boolean variable identified by **inAttrib**. Table 1 lists the GLC boolean variables.

3.5.6 glcGetCallbackFunc

The command

GLCfunc glcGetCallbackFunc (GLCenum inOpcode)

returns the value of the callback function variable identified by **inOpcode**. Table 1 lists the GLC callback function variables.

3.5.7 glcGetListc

The command

const GLCchar* glcGetListc (GLCenum inAttrib, GLint inIndex)

returns the string at offset **inIndex** from the first element in the string list identified by **inAttrib**. Table 1 lists

Name	Enumerant	Initial value	Element count variable
GLC_CATALOG_LIST	0x0080	<empty list>	**GLC_CATALOG_COUNT**

Table 5: String lists

the GLC string lists. The command raises **GLC_PARAMETER_ERROR** if **inIndex** is less than zero or is greater than or equal to the value of the list's element count variable.

3.5.8 glcGetListi

The command

GLint glcGetListi (GLCenum inAttrib, GLint inIndex)

returns the integer at offset **inIndex** from the first element in the integer list identified by **inAttrib**. Table 1

Name	Enumerant	Initial value	Element count variable
GLC_CURRENT_FONT_LIST	0x0090	<empty list>	**GLC_CURRENT_FONT_COUNT**
GLC_FONT_LIST	0x0091	<empty list>	**GLC_FONT_COUNT**
GLC_LIST_OBJECT_LIST	0x0092	<empty list>	**GLC_LIST_OBJECT_COUNT**
GLC_TEXTURE_OBJECT_LIST	0x0093	<empty list>	**GLC_TEXTURE_OBJECT_COUNT**

Table 6: Integer lists

lists the GLC integer lists. The command raises **GLC_PARAMETER_ERROR** if **inIndex** is less than zero or is greater than or equal to the value of the list's element count variable.

3.5.9 glcGetPointer

The command

GLvoid* glcGetPointer (GLCenum inAttrib)

returns the value of the pointer variable identified by **inAttrib**. Table 1 lists the GLC pointer variables.

Name	Enumerant	Initial value
GLC_DATA_POINTER	0x00A0	**GLC_NONE**

Table 7: Pointer variables

3.5.10 glcGetc

The command

const GLCchar* glcGetc (GLCenum inAttrib)

returns the value of the string constant identified by **inAttrib**. Table 8 lists the GLC string constants.

Name	Enumerant
GLC_EXTENSIONS	0x00B0
GLC_RELEASE	0x00B1
GLC_VENDOR	0x00B2

Table 8: String constants

3.5.11 glcGetf

The command

GLfloat glcGetf (GLCenum inAttrib)

returns the value of the floating point variable identified by **inAttrib**. Table 9 lists the GLC floating point

Name	Enumerant	Initial value
GLC_RESOLUTION	0x00C0	0

Table 9: Floating point variables

variables.

3.5.12 glcGetfv

The command

GLfloat* glcGetfv (GLCenum inAttrib, GLfloat* outVec)

stores into **outVec** the value of the floating point vector variable identified by **inAttrib**. If the command does not raise an error, its return value is **outVec**. Table 10 lists the GLC floating point vector variables.

Name	Enumerant	Initial value
GLC_BITMAP_MATRIX	0x00D0	[1 0 0 1]

Table 10: Floating point vector variables

3.5.13 glcGeti

The command

GLint glcGeti (GLCenum inAttrib)

returns the value of the integer variable or constant identified by **inAttrib**. Table 11 lists the GLC integer

Name	Enumerant	Initial value
GLC_CATALOG_COUNT	0x00E0	<implementation specific>
GLC_CURRENT_FONT_COUNT	0x00E1	0
GLC_FONT_COUNT	0x00E2	0
GLC_LIST_OBJECT_COUNT	0x00E3	0
GLC_MASTER_COUNT	0x00E4	<implementation specific>
GLC_MEASURED_CHAR_COUNT	0x00E5	0
GLC_RENDER_STYLE	0x00E6	**GLC_BITMAP**
GLC_REPLACEMENT_CODE	0x00E7	0
GLC_STRING_TYPE	0x00E8	**GLC_UCS1**
GLC_TEXTURE_OBJECT_COUNT	0x00E9	0
GLC_VERSION_MAJOR	0x00EA	<implementation specific>
GLC_VERSION_MINOR	0x00EB	<implementation specific>

Table 11: Integer variables and constants

variables and constants.

3.5.14 glcIsEnabled

The command

GLboolean glcIsEnabled (GLCenum inAttrib)

returns **GL_TRUE** iff the value of the boolean variable identified by **inAttrib** is **GL_TRUE**.

3.5.15 glcStringType

The command

void glcStringType (GLCenum inStringType)

assigns the value **inStringType** to the variable **GLC_STRING_TYPE**. Table 12 lists the GLC string types.

Name	Enumerant
GLC_UCS1	0x0110
GLC_UCS2	0x0111
GLC_UCS4	0x0112

Table 12: String types

3.6 Master commands

Some GLC commands have a parameter **inMaster**. This parameter is an offset from the first element in the GLC master list. The command raises **GLC_PARAMETER_ERROR** if **inMaster** is less than zero or is greater than or equal to the value of the variable **GLC_MASTER_COUNT**.

3.6.1 glcAppendCatalog

The command

void glcAppendCatalog (const GLCchar* inCatalog)

appends the string **inCatalog** to the list **GLC_CATALOG_LIST**.

3.6.2 glcGetMasterListc

The command

const GLCchar* glcGetMasterListc (GLint inMaster, GLCenum inAttrib, GLint inIndex)

returns a string from a string list that is an attribute of the master identified by **inMaster**. The string list is identified by **inAttrib**. The command returns the string at offset **inIndex** from the first element in this string list. Table 1 lists the string list attributes that are associated with each GLC master and font. The command

Name	Enumerant	Element count attribute
GLC_CHAR_LIST	0x0050	**GLC_CHAR_COUNT**
GLC_FACE_LIST	0x0051	**GLC_FACE_COUNT**

Table 13: Master/font string list attributes

raises **GLC_PARAMETER_ERROR** if **inIndex** is less than zero or is greater than or equal to the value of the list's element count attribute.

3.6.3 glcGetMasterMap

The command

const GLCchar* glcGetMasterMap (GLint inMaster, GLint inCode)

returns the string name of the character that the master identified by **inMaster** maps **inCode** to. If the master does not map **inCode**, the command returns **GLC_NONE**.

3.6.4 glcGetMasterc

The command

const GLCchar glcGetMasterc (GLint inMaster, GLCenum inAttrib)

returns a string attribute of the master identified by **inMaster**. The attribute is identified by **inAttrib**. Table 1

Name	Enumerant
GLC_FAMILY	0x0060
GLC_MASTER_FORMAT	0x0061
GLC_VENDOR	0x0062
GLC_VERSION	0x0063

Table 14: Master/font string attributes

lists the string attributes that are associated with each GLC master and font.

3.6.5 glcGetMasteri

The command

GLint glcGetMasteri (GLint inMaster, GLCenum inAttrib)

returns an integer attribute of the master identified by **inMaster**. The attribute is identified by **inAttrib**. Table 15 lists the integer attributes that are associated with each GLC master and font.

Name	Enumerant
GLC_CHAR_COUNT	0x0070
GLC_FACE_COUNT	0x0071
GLC_IS_FIXED_PITCH	0x0072
GLC_MAX_MAPPED_CODE	0x0073
GLC_MIN_MAPPED_CODE	0x0074

Table 15: Master/font integer attributes

3.6.6 glcPrependCatalog

The command

void glcPrependCatalog (const GLCchar* inCatalog)

prepends the string **inCatalog** to the list **GLC_CATALOG_LIST**.

3.6.7 glcRemoveCatalog

The command

void glcRemoveCatalog (GLint inIndex)

removes a string from the list **GLC_CATALOG_LIST**. The command removes the string at offset **inIndex** from the first element in the list. The command raises **GLC_PARAMETER_ERROR** if **inIndex** is less than zero or is greater than or equal to the value of the variable **GLC_CATALOG_COUNT**.

3.7 Font commands

Most of the commands in this category have a parameter **inFont**. Unless otherwise specified, these commands raise **GLC_PARAMETER_ERROR** if **inFont** is not the ID of a font.

3.7.1 glcAppendFont

The command

void glcAppendFont (GLint inFont)

appends **inFont** to the list **GLC_CURRENT_FONT_LIST**. The command raises **GLC_PARAMETER_ERROR** if inFont is an element in the list **GLC_CURRENT_FONT_LIST** at the beginning of command execution.

3.7.2 glcDeleteFont

The command

void glcDeleteFont (GLint inFont)

deletes the font identified by **inFont**. If **inFont** is an element in the list **GLC_CURRENT_FONT_LIST**, the command removes that element from the list.

3.7.3 glcFont

The command

void glcFont (GLint inFont)

begins by removing all elements from the list **GLC_CURRENT_FONT_LIST**. If **inFont** is nonzero, the command then appends **inFont** to the list. Otherwise, the command does not raise an error and the list remains empty.

3.7.4 glcFontFace

The command

GLboolean glcFontFace (GLint inFont, const GLCchar* inFace)

attempts to set the current face of the font identified by **inFont** to the face identified by the string **inFace**. If **inFace** is not an element of the font's string list attribute **GLC_FACE_LIST**, the command leaves the font's current face unchanged and returns **GL_FALSE**. If the command succeeds, it returns **GL_TRUE**.

If **inFont** is zero, the command iterates over the list **GLC_CURRENT_FONT_LIST**. For each of the fonts named therein, the command attempts to set the font's current face to the face in that font that is identified by **inFace**. In this case, the command returns **GL_TRUE** iff **GLC_CURRENT_FONT_LIST** contains one or more elements and the command successfully sets the current face of each of the fonts named in the list.

3.7.5 glcFontMap

The command

void glcFontMap (GLint inFont, GLint inCode, const GLCchar* inCharName)

modifies the map of the font identified by **inFont** such that the font maps **inCode** to the character whose name is the string **inCharName**. If **inCharName** is **GLC_NONE**, **inCode** is removed from the font's map. The command raises **GLC_PARAMETER_ERROR** if **inCharName** is not **GLC_NONE** or an element of the font's string list attribute **GLC_CHAR_LIST**.

3.7.6 glcGenFontID

The command

void glcGenFontID (void)

returns a font ID that is not an element of the list **GLC_FONT_LIST**.

3.7.7 glcGetFontFace

The command

const GLCchar* glcGetFontFace (GLint inFont)

returns the string name of the current face of the font identified by **inFont**.

3.7.8 glcGetFontListc

The command

const GLCchar* glcGetFontListc (GLint inFont, GLCenum inAttrib, GLint inIndex)

is identical to the command **glcGetMasterListc** (Section 3.6.2), except that it operates on a font (identified by **inFont**), not a master.

3.7.9 glcGetFontMap

The command

const GLCchar* glcGetFontMap (GLint inFont, GLint inCode)

is identical to the command **glcGetMasterMap** (Section 3.6.3), except that it operates on a font (identified by **inFont**), not a master.

3.7.10 glcGetFontc

The command

const GLCchar* glcGetFontc (GLint inFont, GLCenum inAttrib)

is identical to the command **glcGetMasterc** (Section 3.6.4), except that it operates on a font (identified by **inFont**), not a master.

3.7.11 glcGetFonti

The command

GLint glcGetFonti (GLint inFont, GLCenum inAttrib)

is identical to the command **glcGetMasteri** (Section 3.6.5), except that it operates on a font (identified by **inFont**), not a master.

3.7.12 glcIsFont

The command

GLboolean glcIsFont (GLint inFont)

returns **GL_TRUE** iff **inFont** is the ID of a font. If **inFont** is not the ID of a font, the command does not raise an error.

3.7.13 glcNewFontFromFamily

The command

GLint glcNewFontFromFamily (GLint inFont, const GLCchar *inFamily)

performs a sequential search beginning with the first element of the GLC master list, looking for the first master whose string attribute **GLC_FAMILY** equals **inFamily**. If there is no such master, the command returns zero. Otherwise, the command creates a new font from the master. The ID of the new font is **inFont**. If the command succeeds, it returns **inFont**. If **inFont** is the ID of a font at the beginning of command execution, the command executes the command **glcDeleteFont (inFont)** before creating the new font.

3.7.14 glcNewFontFromMaster

The command

GLint glcNewFontFromMaster (GLint inFont, GLint inMaster)

creates a new font from the master identified by **inMaster**. The ID of the new font is **inFont**. If the command succeeds, it returns **inFont**. If **inFont** is the ID of a font at the beginning of command execution, the command executes the command **glcDeleteFont (inFont)** before creating the new font.

3.8 Transformation commands

Glyph coordinates are defined in the em coordinate system. When the value of **GLC_RENDER_STYLE** is **GLC_BITMAP**, GLC uses the 2x2 **GLC_BITMAP_MATRIX** to transform layouts from the em coordinate system to the GL raster coordinate system in which bitmaps are drawn. The GLC transformation commands modify the value of **GLC_BITMAP_MATRIX**.

When the value of the variable **GLC_RENDER_STYLE** is not **GLC_BITMAP**, GLC performs no transformations on glyph coordinates. In this case, GLC uses em coordinates directly as GL world coordinates when drawing a layout, and it is the responsibility of the GLC client to issue GL commands that set up the appropriate GL transformations.

3.8.1 glcLoadIdentity

The command

void glcLoadIdentity (void)

assigns the value [1 0 0 1] to the floating point vector variable **GLC_BITMAP_MATRIX**.

3.8.2 glcLoadMatrix

The command

void glcLoadMatrix (const GLfloat *inMatrix)

assigns the value [**inMatrix[0] inMatrix[1] inMatrix[2] inMatrix[3]**] to the floating point vector variable **GLC_BITMAP_MATRIX**.

3.8.3 glcMultMatrix

The command

void glcMultMatrix (const GLfloat *inMatrix)

assigns the value [*a b c d*] to the floating point vector variable **GLC_BITMAP_MATRIX,** where

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} = \begin{bmatrix} \mathbf{GLC\_BITMAP\_MATRIX[0]} & \mathbf{GLC\_BITMAP\_MATRIX[2]} \\ \mathbf{GLC\_BITMAP\_MATRIX[1]} & \mathbf{GLC\_BITMAP\_MATRIX[3]} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{inMatrix[0]} & \mathbf{inMatrix[2]} \\ \mathbf{inMatrix[1]} & \mathbf{inMatrix[3]} \end{bmatrix}$$

3.8.4 glcRotate

The command

void glcMultMatrix (GLfloat inAngle)

assigns the value [*a b c d*] to the floating point vector variable **GLC_BITMAP_MATRIX,** where **inAngle** is measured in degrees, $\theta = \mathbf{inAngle} \cdot \pi / 180$, and

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} = \begin{bmatrix} \mathbf{GLC\_BITMAP\_MATRIX[0]} & \mathbf{GLC\_BITMAP\_MATRIX[2]} \\ \mathbf{GLC\_BITMAP\_MATRIX[1]} & \mathbf{GLC\_BITMAP\_MATRIX[3]} \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

3.8.5 glcScale

The command

void glcMultMatrix (GLfloat inX, GLfloat inY)

assigns the value [*a b c d*] to the floating point vector variable **GLC_BITMAP_MATRIX,** where

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} = \begin{bmatrix} \mathbf{GLC\_BITMAP\_MATRIX[0]} & \mathbf{GLC\_BITMAP\_MATRIX[2]} \\ \mathbf{GLC\_BITMAP\_MATRIX[1]} & \mathbf{GLC\_BITMAP\_MATRIX[3]} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{inX} & 1 \\ 1 & \mathbf{inY} \end{bmatrix}$$

3.9 Rendering commands

This section begins with definitions of helper functions that are not part of the GLC API but are needed to define the semantics of the GLC rendering and measurement commands. The name prefix "__" indicates that these functions are not GLC commands.

The function **GLint __glcLookupFont (GLint inCode)** returns the ID of the first font in **GLC_CURRENT_FONT_LIST** that maps **inCode**. If there is no such font, the function returns zero.

```
static GLint __glcLookupFont ( GLint inCode ) {
    GLint i;
    const GLint n = glcGeti ( GLC_CURRENT_FONT_COUNT );
    for ( i = 0 ; i < n ; ++i ) {
        const GLint font = glcGetListi ( GLC_CURRENT_FONT_LIST, i );
        if ( glcGetFontMap ( font, inCode ) ) return font;
    }
    return 0;
}
```

The function **GLint __glcGetFont (GLint inCode)** returns the ID of the first font in **GLC_CURRENT_FONT_LIST** that maps **inCode**. If there is no such font, the function attempts to append to **GLC_CURRENT_FONT_LIST** and return the ID of a font that maps **inCode**. If the attempt fails, the function returns zero.

If a callback function is defined for **GLC_OP_glcUnmappedCode**, **__glcGetFont** calls the function. The callback function should return **GL_TRUE** iff it succeeds in appending to **GLC_CURRENT_FONT_LIST** the ID of a font that maps **inCode**.

If the value of the boolean variable **GLC_AUTO_FONT** is **GL_TRUE**, **__glcGetFont** searches **GLC_FONT_LIST** for the first font that maps **inCode**. If the search succeeds, **__glcGetFont** appends the font's ID to **GLC_CURRENT_FONT_LIST**. Otherwise, the function searches the GLC master list for the first master that maps **inCode**. If the search succeeds, **__glcGetFont** creates a font from the master and appends the font's ID to **GLC_CURRENT_FONT_LIST**.

```
static GLint __glcGetFont ( GLint inCode ) {
    GLCfunc callbackFunc;
    GLint font;
    if ( font = __glcLookupFont ( inCode ) ) return font;
    callbackFunc = glcGetCallbackFunc ( GLC_OP_glcUnmappedCode );
    if ( callbackFunc && ( *callbackFunc ) ( inCode ) ) {
        if ( font = __glcLookupFont ( inCode ) ) return font;
    }
    if ( glcIsEnabled ( GLC_AUTO_FONT ) ) {
        GLint i, n;
        n = glcGeti ( GLC_FONT_COUNT );
        for ( i = 0 ; i < n ; ++i ) {
            font = glcGetListi ( GLC_FONT_LIST, i );
            if ( glcGetFontMap ( font, inCode ) ) {
                glcAppendFont ( font );
                return font;
            }
        }
        n = glcGeti ( GLC_MASTER_COUNT );
        for ( i = 0 ; i < n ; ++i ) {
            if ( glcGetMasterMap ( i, inCode ) ) {
                font = glcNewFontFromMaster ( glcGenFontID ( ), i );
                if ( font ) {
                    glcAppendFont ( font );
```

```
                    return font;
                }
            }
        }
    }
    return 0;
}
```

The function **void __glcRenderChar (GLint inCode)** creates a layout for the character that **inCode** is mapped to, then issues GL commands to draw the layout.

```
void __glcRenderChar ( GLint inCode ) {
    switch ( glcGeti ( GLC_RENDER_STYLE ) ) {
        case GLC_BITMAP: __glcRenderBitmapChar ( inCode ); break;
        case GLC_LINE: __glcRenderLineChar ( inCode ); break;
        case GLC_TRIANGLE: __glcRenderTriangleChar ( inCode ); break;
        case GLC_TEXTURE: __glcRenderTextureChar ( inCode ); break;
    }
}

void __glcRenderBitmapChar ( GLint inCode ) {
    __glcBitmapLayout *const l = __glcCreateBitmapLayout ( inCode );
    glBitmap ( l->size.x, l->size.y, l->orig.x, l->orig.y, l->advance.x, l->advance.y, l->image );
}

void __glcRenderLineChar ( GLint inCode ) {
    GLint i, j;
    __glcLineLayout *const l = __glcCreateLineLayout ( inCode );
    GLfloat *v = l->vtxCoords;
    for ( i = 0 ; i < l->stripCount ; ++i ) {
        glBegin ( GL_LINE_STRIP );
        for ( j = 0 ; j < l->stripLen[i] ; ++j, v += 2 ) glVertex2fv ( v );
        glEnd ( );
    }
    glTranslatef ( l->advance.x, 0, 0 );
}

void __glcRenderTextureChar ( GLint inCode ) {
    GLint i;
    __glcTextureLayout *const l = __glcCreateTextureLayout ( inCode );
    GLfloat *t = l->texCoords;
    GLfloat *v = l->vtxCoords;
    for ( i = 0 ; i < l->texLevels ; ++i ) {
        glTexImage2D (
            GL_TEXTURE_2D, i, 2, l->tex[i].size.x, l->tex[i].size.y, 0,
            GL_LUMINANCE_ALPHA, GL_UNSIGNED_BYTE, l->tex[i].image
        );
    }
    glBegin ( GL_TRIANGLE_STRIP );
    for ( i = 0 ; i < 4 ; ++i, t += 2, v += 2 ) {
        glTexCoord2fv ( t );
        glVertex2fv ( v );
    }
    glEnd ( );
```

```
    glTranslatef ( l->advance.x, 0, 0 );
}

void __glcRenderTriangleChar ( GLint inCode ) {
    GLint i, j;
    __glcTriangleLayout *const l = __glcCreateTriangleLayout ( inCode );
    GLfloat *v = l->vtxCoords;
    for ( i = 0 ; i < l->stripCount ; ++i ) {
        glBegin( GL_TRIANGLE_STRIP );
        for ( j = 0 ; j < l->stripLen[i] ; ++j, v += 2 ) glVertex2fv ( v );
        glEnd ( );
    }
    glTranslatef ( l->advance.x, 0, 0 );
}
```

The function **void __glcRaiseError (GLCenum inError)** retrieves the value of the issuing thread's GLC error code variable. If the value is **GLC_NONE**, the function assigns the value **inError** to the variable.

3.9.1 glcRenderChar

The command

void glcRenderChar (GLint inCode)

renders the character that **inCode** is mapped to.

If the function **__glcGetFont** cannot find a font that maps **inCode**, the command attempts to produce an alternate rendering. If the value of the variable **GLC_REPLACEMENT_CODE** is nonzero, and **__glcGetFont** finds a font that maps the replacement code, the command renders the character that the replacement code is mapped to. Otherwise, the command attempts to render the character sequence **<***hexcode***>**, where \\ is the character REVERSE SOLIDUS (U+5C), **<** is the character LESS-THAN SIGN (U+3C), **>** is the character GREATER-THAN SIGN (U+3E), and *hexcode* is **inCode** represented as a sequence of hexadecimal digits. The sequence has no leading zeros, and alphabetic digits are in upper case. The GLC measurement commands treat the sequence as a single character.

The command can be implemented as follows. A different implementation is acceptable iff its externally observable behavior is identical.

```
void glcRenderChar ( GLint inCode ) {
    GLint repCode;
    if ( !glcGetCurrentContext ( ) ) {
        __glcRaiseError ( GLC_STATE_ERROR );
        return;
    }
    if ( __glcGetFont ( inCode ) ) {
        __glcRenderChar ( inCode );
        return;
    }
    repCode = glcGeti ( GLC_REPLACEMENT_CODE );
    if ( repCode && __glcGetFont ( repCode ) ) {
        __glcRenderChar ( repCode );
    } else {
        char buf[10];
        GLint i, n;
        if (!__glcGetFont('\\') || !__glcGetFont ( '<' ) || !__glcGetFont ( '>' ) ) return;
```

```
            sprintf ( buf, "%X", inCode );
            n = strlen ( buf );
            for ( i = 0 ; i < n ; ++i ) if ( !__glcGetFont ( buf[i] ) ) return;
            __glcRenderChar('\\');
            __glcRenderChar ( '<' );
            for ( i = 0 ; i < n ; ++i ) __glcRenderChar ( buf[i] );
            __glcRenderChar ( '>' );
        }
    }
```

3.9.2 glcRenderCountedString

The command

void glcRenderCountedString (GLint inCount, const GLCchar *inString)

is identical to the command **glcRenderChar** (Section 3.9.1), except that it renders a string of characters. The string comprises the first **inCount** elements of the array **inString**, which need not be followed by a zero element. The command raises **GLC_PARAMETER_ERROR** if **inCount** is less than zero.

3.9.3 glcRenderString

The command

void glcRenderString (const GLCchar *inString)

is identical to the command **glcRenderCountedString** (Section 3.9.2), except that **inString** is zero terminated, not counted.

3.9.4 glcRenderStyle

The command

void glcRenderStyle (GLCenum inStyle)

assigns the value **inStyle** to the variable **GLC_RENDER_STYLE**. Table 16 lists the GLC rendering styles.

Name	Enumerant
GLC_BITMAP	0x0100
GLC_LINE	0x0101
GLC_TEXTURE	0x0102
GLC_TRIANGLE	0x0103

Table 16: Rendering styles

3.9.5 glcReplacementCode

The command

void glcReplacementCode (GLint inCode)

assigns the value **inCode** to the variable **GLC_REPLACEMENT_CODE**.

3.9.6 glcResolution

The command

void glcResolution (GLfloat inVal)

assigns the value **inVal** to the variable **GLC_RESOLUTION**.

3.10 Measurement commands

Table 17 lists the GLC metrics for character and string layouts. **GLC_BASELINE** is the line segment from

Name	Enumerant	Vector
GLC_BASELINE	0x0030	[$x_l\ y_l\ x_r\ y_r$]
GLC_BOUNDS	0x0031	[$x_{lb}\ y_{lb}\ x_{rb}\ y_{rb}\ x_{rt}\ y_{rt}\ x_{lt}\ y_{lt}$]

Table 17: Metrics

the origin of the layout to the origin of the following layout. **GLC_BOUNDS** is the bounding box of the layout. Figure 2 illustrates the baseline and bounds of a layout.

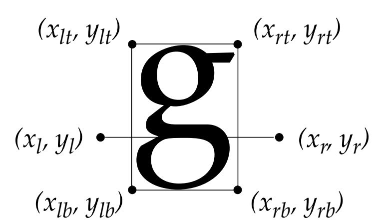

Figure 2: Baseline and bounds

Each point (x, y) is computed in em coordinates, with the origin of a layout at (0, 0). Iff the value of the variable **GLC_RENDER_STYLE** is **GLC_BITMAP**, each point is transformed by the 2x2 **GLC_BITMAP_MATRIX**.

3.10.1 glcGetCharMetric

The command

GLfloat* glcGetCharMetric (GLint inCode, GLCenum inMetric, GLfloat *outVec)

is identical to the command **glcRenderChar**, except that instead of rendering the character that **inCode** is mapped to, the command measures the resulting layout and stores in **outVec** the value of the metric identified by **inMetric**. Table 17 lists the GLC metrics. If the command does not raise an error, its return value is **outVec**.

3.10.2 glcGetMaxCharMetric

The command

GLfloat* glcGetMaxCharMetric (GLCenum inMetric, GLfloat *outVec)

measures the layout that would result from rendering all mapped characters at the same origin, as follows:

```
GLint code;
for ( code = 0 ; code <= 0x7fffffff ; ++code ) {
    glPushAttrib ( GL_CURRENT_BIT );
    glPushMatrix ( );
    if ( __glcLookupFont ( code ) ) glcRenderChar ( code );
    glPopAttrib ( );
    glPopMatrix ( );
}
```

The command stores in **outVec** the value of the metric identified by **inMetric**. Table 17 lists the GLC metrics. If the command does not raise an error, its return value is **outVec**.

3.10.3 glcGetStringCharMetric

The command

GLfloat* glcGetStringCharMetric (GLint inIndex, GLCenum inMetric, GLfloat *outVec)

retrieves a character metric from the GLC measurement buffer and stores it in **outVec**. The character is identified by **inIndex**, and the metric is identified by **inMetric**. Table 17 lists the GLC metrics. The command raises **GLC_PARAMETER_ERROR** if **inIndex** is less than zero or is greater than or equal to the value of the variable **GLC_MEASURED_CHAR_COUNT**. If the command does not raise an error, its return value is **outVec**.

3.10.4 glcGetStringMetric

The command

GLfloat* glcGetStringMetric (GLCenum inMetric, GLfloat *outVec)

retrieves a string metric from the GLC measurement buffer and stores it in **outVec**. The metric is identified by **inMetric**. Table 17 lists the GLC metrics. If the command does not raise an error, its return value is **outVec**.

3.10.5 glcMeasureCountedString

The command

```
GLint glcMeasureCountedString (
    GLboolean inMeasureChars, GLint inCount, const GLCchar *inString
)
```

is identical to the command **glcRenderCountedString**, except that instead of rendering a string, the command measures the resulting layout and stores the measurements in the GLC measurement buffer. The string comprises the first **inCount** elements of the array **inString**, which need not be followed by a zero element. The command raises **GLC_PARAMETER_ERROR** if **inCount** is less than zero.

If the value of **inMeasureChars** is nonzero, the command computes metrics for each character and for the overall string, and it assigns the value **inCount** to the variable **GLC_MEASURED_CHARACTER_COUNT**. Otherwise, the command computes metrics only for the overall string, and it assigns the value zero to the variable **GLC_MEASURED_CHARACTER_COUNT**.

If the command does not raise an error, its return value is the value of the variable **GLC_MEASURED_CHARACTER_COUNT**.

3.10.6 glcMeasureString

The command

GLint glcMeasureString (GLboolean inMeasureChars, const GLCchar *inString)

is identical to the command **glcMeasureCountedString** (Section 3.10.5), except that **inString** is zero terminated, not counted.

4 Extensions

Section under construction.

References

1. Segal, Mark, and Akeley, Kurt. The OpenGL™ Graphics System: A Specification (Version 1.0). OpenGL Architectural Review Board. Unpublished. 1995.

2. Microsoft Corp. TrueType 1.0 Font Files: Technical Specification. Revision 1.5. January 1994. Available from Microsoft Corp., Redmond, WA USA.

3. Adobe Systems Incorporated. Adobe Type 1 Font Format. Version 1.1. Addison-Wesley. 1990. ISBN 0-201-57044-0.

4. ISO/IEC 10646-1:1993. International Standard -- Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane.

5. The Unicode Consortium. The Unicode Standard: Worldwide Character Encoding. Version 1.0, Volume 1. Addison-Wesley. 1991. ISBN 0-201-56788-1.

6. The Unicode Consortium. The Unicode Standard: Worldwide Character Encoding. Version 1.0, Volume 2. Addison-Wesley. 1992. ISBN 0-201-60845-6.

7. The Unicode Consortium. Unicode Technical Report #4: The Unicode Standard, Version 1.1. Prepublication edition. 1993. Available from The Unicode Consortium, 1965 Charleston Road, Mountain View, CA 94043 USA.

8. Dunwoody, Craig. The OpenGL™ Stream Codec: A Specification. In preparation. 1996.