OpenGL VIII: wglUseFontOutlines

Dale Rogerson
Microsoft Developer Network Technology Group

July 31, 1995

Click to open or copy the files in the EasyFont sample application for this technical article.

Click to open or copy the files in the GLlib.DLL for this technical article.

Abstract

This article explains how to use the Win32® **wglUseFontOutlines** function. This function creates three-dimensional (3-D) characters based on a TrueType® font for use in OpenGL™-rendered scenes. The EasyFont sample application demonstrates using **wglUseFontOutlines**.

Introduction

I can remember as a kid daydreaming over the big Sears Christmas Wish Book. The object of these dreams was Duke, a plastic German Shepherd companion for the old GI Joe. Duke had a wagon he could pull, complete with a flashing light and siren and a fire hose for fighting forest fires. Best of all, Duke had a rope slide. You would set the string up, hook Duke's mouth on the slide, let go, and off Duke would go to save the forests from being burned down by GI Joe. At least, that was what was *supposed* to happen. What *really* happened was that Duke fell off the slide and broke his tail, which wouldn't stay attached no matter what the advertisements for Crazy Glue claimed.

The point is, pretty early in life we learn that things don't generally meet our expectations. I don't know if I have lowered all my expectations as a result, but the Win32® **wglUseFontOutlines** function certainly exceeded them.

wglUseFontOutlines is a "wiggle" function (named after the **wgl** prefix to its name). A wiggle function is a Win32 function that bridges the gap between the portable OpenGL™ graphics library and the Microsoft Windows® platform. **wglUseFontOutlines** carries TrueType® fonts across this bridge, extruding them into three-dimensional (3-D) objects in the process.

What does **wglUseFontOutlines** do? Basically, **wglUseFontOutlines** takes a TrueType font and creates a display list for each glyph in the font. (A *glyph* is the image of a single character in the font.) **wglUseFontOutlines** extrudes the glyphs in the process of building the display lists. The resulting display lists are used to render 3-D versions of the glyphs on the screen. The display lists can be rotated, translated, texture-mapped, and anything else that is possible in OpenGL.

If you want to display 3-D versions of your favorite TrueType fonts on the screen, **wglUseFontOutlines** will meet or exceed your needs. However, if you need to modify the glyph shapes or know the exact locations of the outline of each glyph (maybe for advanced kerning), **wglUseFontOutlines** will not work for you.

This article describes the parameters to **wglUseFontOutlines** and how to use them.

Parameters

Following is the prototype for **wglUseFontOutlines**:

```
BOOL wglUseFontOutlines(
    HDC  hdc,
    DWORD  first,
    DWORD  count,
    DWORD  listBase,
    FLOAT  deviation,
    FLOAT  extrusion,
    int  format,
    LPGLYPHMETRICSFLOAT lpgmf
```

```
);
```

The following table gives a quick overview of the meaning of each of these parameters.

hdc	Specifies the device context containing the desired outline font. The outline font selected in *hdc* is used to create the display lists in the current rendering context.
first	Specifies the index of the glyph in the font from which to start generating display lists.
count	Specifies the number of glyphs for which to generate display lists.
listBase	Specifies the starting display list.
deviation	Specifies how closely the display lists follow the glyph outline. The higher the value, the rougher the resulting display-list representation. The value of *deviation* must be equal to or greater than zero.
extrusion	Specifies the amount a font is extruded in the negative *z* direction. The value must be equal to or greater than zero. When *extrusion* equals zero, the display lists are not extruded.
format	Specifies the format (either WGL_FONT_LINES or WGL_FONT_POLYGONS) to use in the display lists. When *format* is WGL_FONT_LINES, **wglUseFontOutlines** creates fonts with line segments; when *format* is WGL_FONT_POLYGONS, **wglUseFontOutlines** creates fonts with polygons.
lpgmf	Points to an array of *count* **GLYPHMETRICSFLOAT** structures that is to receive the metrics of the glyphs. When *lpgmf* is NULL, no glyph metrics are returned.

We'll cover all of these parameters in more detail later in the article.

EasyFont Sample Application

The EasyFont sample application demonstrates how to use **wglUseFontOutlines**. The application displays two strings using display lists generated by **wglUseFontOutlines** from TrueType fonts. You can rotate the strings by using the menu items in the Rotate menu. Use the Change menu item under the Option menu to change the string, font, deviation, extrusion, and format that **wglUseFontOutlines** uses to generate the upper string.

Hey, kids! For something really neat, try out the WingDings® font!

Quick Start

For those who like to see quick results, here is some code:

```
// Pixel format and rendering context already set up and initialized.

// Get the DC for the current rendering context.
HDC hdc = wglGetCurrentDC();
CDC* pdc = CDC::FromHandle(hdc) ;

// Select the font you want to use.
CFont* pOldFont = (CFont*)pdc->SelectObject(m_pFontSelected) ;

// Allocate structure for character metrics.
GLYPHMETRICSFLOAT agmf;

// Build the display lists.
wglUseFontOutlines(hdc,               // DC with font
                   __toascii('A'),    // Character to generate
                   1,                 // Number of characters
                   1000,              // Display-list number
                   0.0,               // Deviation
                   0.1,               // Extrusion
                   WGL_FONT_POLYGONS, // Format
                   &agmf) ;           // Metrics pointer

// Set up transformation.
glLoadIdentity();
```

```
glTranslated(0.5, 0.0, -2.0);

// Display the 3-D letter 'A'
glCallList(1000) ;
```

In the code listed above, **wglUseFontOutlines** used the glyph for the letter *A* (index 65) to generate the display list numbered 1000. Display list 1000 contains an extruded representation of the letter *A* built with polygons. The letter is displayed using the **glCallList** function.

The first step in using **wglUseFontOutlines** is to create a TrueType font and select it into the device context. Much has already been written about choosing and creating fonts for Windows, so I will just refer you to the technical articles listed in the bibliography.

Now that you have seen a simple example, we can examine the parameters of **wglUseFontOutlines** in more detail.

Display Lists

The first parameters we are going to look at are *count* and *listBase*. It is unlikely that you will only need one glyph from the font; in most cases you will need all or most of the glyphs in a font. The following code will get all 256 glyphs in an 8-bit font:

```
// Allocate structure for character metrics.
GLYPHMETRICSFLOAT agmf[256];

// Build the display lists.
wglUseFontOutlines(hdc,               // DC with font
                   0,                 // Starting character index
                   256,               // Number of characters
                   1000,              // Display-list number
                   0.0,               // Deviation
                   0.1,               // Extrusion
                   WGL_FONT_POLYGONS, // Format
                   agmf) ;            // Metrics pointer

// Set up transformation.
glLoadIdentity();
glTranslated(0.5, 0.0, -2.0);

// Display the 3-D letters 'Hello!'
glCallList(1000 + __toascii('H')) ;
glCallList(1000 + __toascii('e')) ;
glCallList(1000 + __toascii('l')) ;
glCallList(1000 + __toascii('l')) ;
glCallList(1000 + __toascii('o')) ;
glCallList(1000 + __toascii('!')) ;
```

In this case, we passed 256 to **wglUseFontOutlines** to generate display lists for all the glyphs in the font. The display list for the first glyph is 1000 and for the 66th glyph (*A*) it is 1065.

Notice that we didn't have to call **glTranslated** between each call to **glCallList**. **wglUseFontOutlines** generates display lists that include positioning commands so that successively called display lists are shown next to each other. How about that for exceeding expectations?

If you think there has to be a better way than calling **glCallList** for each letter you want to display, you are right. You can use **glCallLists**. Just replace the above six **glCallList** statements with the following two statements:

```
glListBase(1000) ;
glCallLists(5, GL_UNSIGNED_BYTE, "Hello!") ;
```

glCallLists uses the third parameter as an array of indexes that are added to the base index set by **glListBase**. Pretty clever, huh?

If you are like me, you don't often want or need to display the ASCII control characters with indexes 0 to 32. Yet the above example has not only created display lists for these characters, it has also generated display lists for the characters above index 127. Because most Windows fonts don't contain any meaningful characters above index 127, this leads to a lot of wasted memory and time. For this reason, I like to use the following code:

```
int iStart = __ascii('!') ;
```

```
int iLast = 126 ;
int iCount = iLast - iStart + 1;

// Allocate structure for character metrics.
GLYPHMETRICSFLOAT agmf[iCount];

// Build the display lists.
wglUseFontOutlines(hdc,               // DC with font
                   iStart,            // Starting character index
                   iCount,            // Number of characters
                   1000,              // Display-list number
                   0.0,               // Deviation
                   0.1,               // Extrusion
                   WGL_FONT_POLYGONS, // Format
                   agmf) ;            // Metrics pointer

// Set up transformation.
glLoadIdentity();
glTranslated(0.5, 0.0, -2.0);

// Display the 3-D letter 'H'
glListBase(1000 - iStart) ;
glCallLists(5, GL_UNSIGNED_BYTE, "Hello!") ;
```

The above code doesn't generate display lists for the first 32 or the last 128 glyphs in the font. To ensure that our trick of passing the string we want to display to **glCallLists** still works, we adjust the base index we pass to **glListBase** by subtracting the index of the first character. This technique should save time and memory. However, developers writing international applications are going to have their work cut out for them.

glGenLists

Although the above code does work, there is a much safer method for working with display lists. In the code above, 1000 is used to start the display list. It is possible that somewhere else in the program there could already be a display list in the range 1000 to 1000 + **iCount**. The code above would erase that existing display list and replace it with a glyph. This isn't much of a problem with a simple program like EasyFont, but could be difficult to debug in a more complicated program.

The solution is to use **glGenLists**, which generates a consecutive range of empty display lists. The code below shows how **CScene::Create3DFont** uses **glGenLists** to get a valid range of display lists. Although not required, it demonstrates the use of **glDeleteLists** to free a consecutive range of display lists.

```
void CScene::Create3DFont()
{
   // Delete existing display-list numbers.
   if (m_iDisplayListStart != 0)
   {
      // Delete the existing lists.

      glDeleteLists(m_iDisplayListStart, m_iNumberChars) ;

   }

   // Generate new display lists.

   m_iDisplayListStart = glGenLists(m_iNumberChars) ;

   m_iDisplayListBase = m_iDisplayListStart - m_iFirstChar ;

   // Get the DC for the current rendering context.
   HDC hdc = wglGetCurrentDC();
   CDC* pdc = CDC::FromHandle(hdc) ;

   // Select the font.
   CFont* pOldFont = (CFont*)pdc->SelectObject(m_pFontSelected) ;

   // Generate the display lists.
   GLYPHMETRICSFLOAT* agmf = new GLYPHMETRICSFLOAT[m_iNumberChars];

   BOOL bResult =
   wglUseFontOutlines(hdc,               // DC with font
                      m_iFirstChar,      // First character
                      m_iNumberChars,    // Number

                      m_iDisplayListStart,// Starting display
                      m_fDeviation,      // Deviation
                      m_fExtrusion,      // Extrusion
                      m_iFormat,         // Format or WGL_FONT_LINES
                      agmf) ;            // Information pointer

.
.
.
```

```
}
```

Deviation

The next parameter to **wglUseFontOutlines** is *deviation*. "Deviation" specifies how closely **wglUseFontOutlines** follows the outlines in the glyphs. A deviation of 0.0 means that **wglUseFontOutlines** follows the outline exactly. Using higher values makes the curves in the fonts progressively less smooth. For example, the following is a smooth TrueType *O*:

With a deviation of 0.5, this would look approximately like:

As you can see, it would be much rougher. The best way to learn about deviation is to play with the EasyFont sample application and change the values for deviation.

Extrusion

The *extrusion* parameter controls how much depth the characters will have. A larger value generates more depth. Again, the best way to learn about this parameter is to play with the sample application.

Format

wglUseFontOutlines can generate display lists with two different formats: WGL_FONT_POLYGONS or WGL_FONT_LINES. When WGL_FONT_POLYGONS is specified, the display lists contain polygons to build solid 3-D versions of the glyphs. If WGL_FONT_LINES is used, the display lists contain line segments that build 3-D outlines of the glyphs.

GLYPHMETRICSFLOAT

In the sample application, I rotate the string "MSDN" around a vertical axis between the *S* and the *D*. Because I also wanted the string to be in the center of the display, I had to translate the starting point for the *M* the same distance down the negative x axis as the width of the *M* and the *S*. To do this, I needed to know how wide the *M* and the *S* were. The GLYPHMETRICFLOAT structure contains the dimensions I needed.

For each display list that **wglUseFontOutlines** creates, it fills in a GLYPHMETRICFLOAT structure. This structure has two fields, **gmfBlackBoxX** and **gmfBlackBoxY**, which contain the smallest rectangle that completely encloses the glyph. The origin of the upper left corner of the enclosing rectangle is contained in **gmfptGlyphOrigin**. However, these were not the fields I needed.

GLYPHMETRICFLOAT also contains two other fields, **gmfCellIncX** and **gmfCellIncY**, which specify the distance from the origin of the current character cell to the origin of the next character cell. **gmfCellIncY** is zero for most European languages. **gmfCellIncX** was the parameter I needed to properly translate the display lists so that the string was centered. This is the code:

```
for (int i = 0 ; i < MSDN_COUNT ; i ++)
{
```

```
   wglUseFontOutlines(hdc,
                      __toascii(MSDN_STRING[i]),
                      1,
                      MSDN_DISPLAYLIST+i,
                      0.0f,
                      0.1f,
                      WGL_FONT_POLYGONS,
                      &m_agmfMSDN[i]) ;

}

// Center the string on the screen.
double scale = 0.5 ;

double widthMN = (m_agmfMSDN[0].gmfCellIncX +

                  m_agmfMSDN[0].gmfCellIncX)* scale;


glLoadIdentity();
glTranslated(0.0, -0.65, -3.0);
glRotated(m_angle.cy, 0.0, 1.0, 0.0);
glTranslated(-widthMN, 0.0, 0.0) ;
glScaled(scale, scale, scale) ;

glColor3ubv(byPink) ;
glCallList(MSDN_DISPLAYLIST) ;
glColor3ubv(byTeal) ;
glCallList(MSDN_DISPLAYLIST+1) ;
glColor3ubv(byBlue) ;
glCallList(MSDN_DISPLAYLIST+2) ;
glColor3ubv(byGreen) ;
glCallList(MSDN_DISPLAYLIST+3) ;
```

Conclusion

wglUseFontOutlines makes adding 3-D fonts to your OpenGL applications easy. Because **wglUseFontOutlines** generates the 3-D fonts from existing TrueType fonts, you don't have to design new fonts and your application can take advantage of any TrueType font already installed on the system. Unlike Duke, **wglUseFontOutlines** exceeded my expectations. Unless you want to do transformations that require the vertices contained in the display lists or need more information than is provided in the GLYPHMETRICFLOAT structure, **wglUseFontOutlines** should also meet your expectations.

Bibliography

Sources of Information on OpenGL

Crain, Dennis. "Windows NT OpenGL: Getting Started." April 1994. (MSDN Library, Technical Articles, Platform Articles, GDI and OpenGL)

Microsoft Win32 Software Development Kit (SDK) for Windows NT 3.5 *OpenGL Programmer's Reference*.

Neider, Jackie, Tom Davis, and Mason Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Release 1* . Reading, MA: Addison-Wesley, 1993. (This book is also known as the "Red Book.")

OpenGL Architecture Review Board. *OpenGL Reference Manual: The Official Reference Document for OpenGL, Release 1* . Reading, MA: Addison-Wesley, 1992. (This book is also known as the "Blue Book.")

Prosise, Jeff. "Advanced 3-D Graphics for Windows NT 3.5: Introducing the OpenGL Interface, Part I." *Microsoft Systems Journal* (October 1994, Volume 9, Number 10). (MSDN Library Archive Edition, Books and Periodicals)

Prosise, Jeff. "Advanced 3-D Graphics for Windows NT 3.5: The OpenGL Interface, Part II." *Microsoft Systems Journal* (November 1994, Volume 9, Number 11). (MSDN Library Archive Edition, Books and Periodicals)

Prosise, Jeff. "Understanding Modelview Transformations in OpenGL for Windows NT." *Microsoft Systems Journal* (February 1995, Volume 10, Number 2).

Rogerson, Dale. "OpenGL I: Quick Start." December 1994. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL II: Windows Palettes in RGBA Mode." December 1994. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL III: Building an OpenGL C++ Class." January 1995. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL IV: Color Index Mode." January 1995. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL V: Translating Windows DIBs." January 1995. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL VI: Rendering on DIBs with PFD_DRAW_TO_BITMAP." April 1995. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL VII: Scratching the Surface of Texture Mapping." April 1995. (MSDN Library, Technical Articles)

Sources of Information on TrueType Fonts

Gery, Ron. "Using TrueType." (MSDN Library Archive Edition, Technical Articles, Windows Articles, GDI and OpenGL Articles, Windows [16-bit] Only)

Gery, Ron. "Windows Font Mapping." (MSDN Library, Technical Articles, Windows Articles, GDI and OpenGL Articles)

Gery, Ron. "GDI Objects." (MSDN Library, Technical Articles, Platform Articles, GDI and OpenGL Articles)

Weise, David, and Dennis Adler. "TrueType and Microsoft Windows Version 3.1." (MSDN Library Archive Edition, Technical Articles, Platform Articles, GDI and OpenGL Articles)