THE DISPLAY OF CHINESE AND ENGLISH CHARACTERS BASED ON OPENGL ES IN SYMBIAN OPERATION SYSTEM

Liu Jinping [a,b,*] Liu Zhengjun [a] Huang Ying [b]

[a] The Chinese Academy of Surveying and Mapping, Beijing 100039-liujinping123321@163.com;
[b] Faculty of Mechanical & Electronic Information, China University of Geosciences, Wuhan 430074-liujinping123321@163.com;

KEY WORDS: Symbian OS,OpenGL ES,Unicode,Chinese display,FreeType,3D

ABSTRACT:

This article is about the method of the display of Chinese and English Characaters in the programming environment of Carbide.c++. Development of this method is based on Symbian Series 60 platform and OpenGL ES. Both Chinese characters and English characters can be displayed in 3D scene. The methodology and arithmetic are presented in the article. This method has been successfully applied to the 3D navigation system of cellphone on Series 60 platform such as Nokia N95. Most city names and typical ground objects can be displayed in our developing system. Moreover, all the names can be displayed either in Chinese or English characters. OpenGL ES application programming interface doesn't provide the ready methods of drawing or editing text. FreeType 2 application programming interface provides us the methods of drawing the character's glyph outline according to the character's glyph index in some coding methods. In this paper, the characters can be displayed by converting the coding method of Symbian characters to Unicode coding method and also the methods of obtaining bitmaps of characters provided by FreeType 2 application programming interface is realizable. ANSI (American National Standards Institute) coding method is the default coding method of the compiler in Symbian operating system. The ANSI coding method of Chinese characters in the Symbian operating compiler is actual GBK coding method. All the characters will be converted to Unicode characters before being converted to glyph index in the FreeType 2 application programming interface. The characters with coding methods are ANSI or GBK which are stored in characters buffer, then the whole character string in character buffer is converted to Unicode string on Series 60 platform. No FreeType 2 application programming interface in Symbian Series 60 SDK. C programming language is transplanted to the Symbian operating system which is developed in C++ programming language. At the meanwhile, FreeType 2 application programming interface is transplanted to Series 60 platform. Then we use the methods provided by the static library of FreeType 2 to acquire these characters' glyph index and glyph images in the 3D navigation application. Finally, we apply these glyph images as textures of the model to truly realize the 3D display of these Chinese and English characters. The method of drawing or editing text in OpenGL ES has strong transplanted while it is a convenient method for drawing text in OpenGL ES. We can adopt this method to draw many kinds of text in various applications based on OpenGL ES or OpenGL. The character font and color can be changed in the actual programming environment.

1. INTRODUCTION

Lifelike 3D scene correctly simulates the reality environment in 3D navigation system. The city names and typical ground objects' names in 3D navigation scene bring our eyes intuitionistic visual effect and quick resolving power. They play an important role in 3D navigation scene. It is necessary to label these cities and typical ground objects in 3D scene.

The stroke character and raster character are the two character's forms in OpenGL. The raster function interface has been wiped off in OpenGL ES. The outline of stroke character is drawn and added as elementary geometric graphic unit to graphics system in the form of stroke character. If all the characters in ASCII format are defined as geometric graphic units, these geometric graphic units will cost the device plenty of memory and much treatment time. Considering the limited memory of the cellphone, the total of the Chinese characters is too large to define the majority of the Chinese characters which are in common use in our daily life as the geometric graphic units. Using FreeType 2 API to draw the outline of Chinese and English characters which need be shown in screen and converting their outlines to bitmaps which are saved as the textures of character models give much convenience to display Chinese and English characters' models in the screen of cellphone.

* Corresponding author. Liujinping123321@163.com

2. METHODOLOGY

OpenGL ES API and FreeType 2 API provide the key methods to realize the characters' display in 3D scene.

2.1 The Summarization of OpenGL ES and FreeType2

2.1.1 OpenGL ES: It is specially designed for cellphone and

other embedded device. OpenGL ES API provides the rock-bottom functions of display and the standard methods to describe the rock-bottom hardware of rendering image in main CPU or graphics processing unit. As a given 3D application programming interface of cellphone, OpenGL ES allow programmers adding the 3D graphic accelerator to the cellphone in a standard mode. When the scheduled 3D graphics is displayed in screen, making use of graphical interfaces can realize the various transformations and many settings in 3D display.

2.1.2 FreeType 2 API: FreeType 2 engine can efficiently produce transplantable symbol image with high quality and only cost a small quantity of memory. It provides a set of uniform application programming interface which is easy to use and independent on the type of font file. FreeType 2 API supports scaleable typeface such as TrueType and can return the outline of font to the client programs.

2.2 Methodology of Displaying Character

Freetype 2 API can convert the characters to glyph images. Applying these glyph images as the textures of models truly realize the 3D display of these Chinese and English characters in 3D scene. The flow chart of displaying characters is shown as Figure 1.

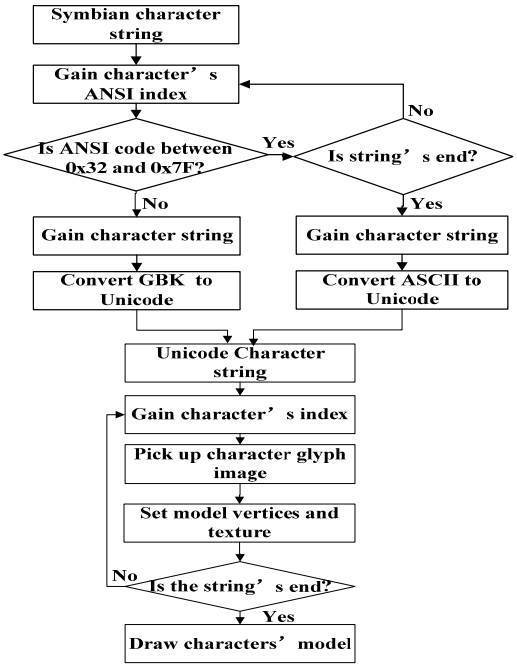

Figure 1.Flow chart of character string display

2.2.1 Coding Conversion: Different countries make different standards for coding. ANSI character set is the extending of ASCII. It contains ASCII character set and the character sets which derive from ASCII character set and are compatible with ASCII character set. ANSI codes English characters with single byte and double bytes for Chinese characters. The default Chinese coding method of text is GBK in the situation of compiling program in Windows. ANSI coding is actual GB2312 coding in simplified Chinese character system. The standard of coding with different bytes in English characters and other lingual characters makes various characters correctly processed in local computer system but wrong in non-local computer system. The default coding method in Symbian OS is Unicode. Unicode contains all the coding schemes of the worldwide languages and almost designs uniform and exclusive binary code for every character in any language. The development tool which we use in Windows is Carbide.c++. Converting Chinese character coding from ANSI to Unicode before displaying Chinese character is obligatory in Symbian OS.

Character string is end with '\0' (0x00) in C program. The disposal in computers considers 0x00 is the end of character string. One byte is 0x00 and the other is not in many codes which are coded with two bytes in Unicode coding method. When these Unicode codes are disposed in C program compiler, they will cause error or confusion in disposal. One solution is using the types of wchar_t to express a Unicode character variable. The variable type wchar_t make computers read two bytes every time and end with 0x0000.

We input the city names or typical ground objects' names to the character descriptor in English in application:

```
_LIT8 (KPlaceName, "Shanghai");
```

"Shanghai" is saved as ASCII coding in Windows. When all the characters' ASCII coding value is between 0x32 and 0x7F, we can directly convert character string from multi-byte string to a wide character string by the follow codes:

```
TBuf8<40> ibuf(KPlaceName);
const char *charString = (const char*)ibuf.PtrZ();
wchar_t* chstr=(wchar_t*)malloc((ibuf.Length()+1)
*sizeof(wchar_t));
TInt ret=mbstowcs(chstr, charString,ibuf.Length() );
chstr[ibuf.Length()]='\0';
```

Application will gain glyph index of each character in character string chstr from font file according to the character's Unicode code. When we input city names in Chinese:

```
_LIT8(KPlaceName,"上海");
```

"上海" is saved with GBK code. Simplified Chinese character coding method is GB2312 or GBK in program whose source file is saved as non-UTF-8 codes. We need convert the coding method from ANSI to Unicode before handling the character string in Symbian OS. "上" and "海" are respective saved in double bytes as [0xBA,0xA3] and [oxC9,0xCA] in ANSI coding method and saved as 0x4E0A and 0x6D77 in Unicode coding method. The code snippet of converting Chinese character string coding method from GBK to Unicode is shown as follows:

```
TBuf8<40> ibuf(KPlaceName);
char* chstr=(char*)ibuf.PtrZ();
 CCnvCharacterSetConverter*
converter=CCnvCharacterSetConverter::NewL();
if (converter->PrepareToConvertToOrFromL
(KCharacterSetIdentifierGbk,
```

```
CEikonEnv::Static()->FsSession())
  == CCnvCharacterSetConverter::EAvailable) {}
else if (converter->PrepareToConvertToOrFromL
(KCharacterSetIdentifierGb2312,
CEikonEnv::Static()->FsSession())!=CCnvCharacterSetConver
ter::EAvailable)
  {
CleanupStack::PopAndDestroy();
User::Leave(KErrNotSupported);
  }
TText8 *tstr = (TText8*)chstr;
TInt state=CCnvCharacterSetConverter::KStateDefault;
TPtrC8 source(tstr);
HBufC* iInfoText =HBufC::NewL(source.Length());
TPtr16 ptr = iInfoText->Des();
if (CCnvCharacterSetConverter::EErrorIllFormedInput==
converter->ConvertToUnicode(ptr, source, state) )
{
  CleanupStack::PopAndDestroy();
User::Leave(KErrArgument);
}
wchar_t* wstr=(wchar_t*)ptr.PtrZ();
```

"上海" is saved with Unicode coding method in wstr. The transformation above is also effective to mixed characters inputting with English and Chinese characters.

2.2.2 Load FreeType 2 Library: No FreeType 2 application programming interface in Symbian Series 60 SDK. We transplant the open source code of FreeType 2 to Series 60 platform and build Freetype static library. Some functions without relation to produce glyph image in FreeType 2 API are wiped off before building static FreeType 2 library in order to reducing the size of static FreeType 2 library and the time of running the static library. The simplified FreeType 2 library is loaded to Series 60 platform. It provides us all-sided methods to acquire character's glyph index and glyph image in the 3D navigation application.

2.2.3 Generate Character's Bitmap: An new instance of FreeType 2 API is created and a font file is loaded at first. Font is one group of characters' image which can be shown and printed. One type of font defines one mode of a character set for a series of given characters. The font file includes some tables by the name of character image for converting character code to glyph index. One font file contains a set of glyph. Every glyph can be saved as bitmap or some other structures. Font file saves glyph in arbitrary order. We access a character's glyph by its glyph index.

FreeType 2 API uses size object to construct the given character's size in given font face. When a new font face is built, the default size is 10 pixels both in horizontal direction and vertical direction in scalable typeface. The size must be set before loading a glyph. Unicode character image is the default character image when we build a new font face. If the font doesn't contain the Unicode character image, FreeType 2 will attempt to simulate a new one.

We load glyph image according to the relevant glyph index. The glyph with different font is saved in different format. Every glyph image is a bitmap in typeface of fixed size. It describes a glyph with vector form namely outline in scaleable typeface, such as TrueType and Type1. A TrueType font file simkai.ttf is loaded to the application. The way to gain all the characters' glyph images of a label is realized by creating a loop in application. Each trip through the loop loads a glyph image. We transform the glyph image to an anti-aliased bitmap which is used as a texture for character's model. Then we set the model's width and height according to the left and top span of character outline and the width and height of the bitmap. The process of gaining character outline and bitmap is important. The link of generating glyph bitmaps is shown in Figure 2.

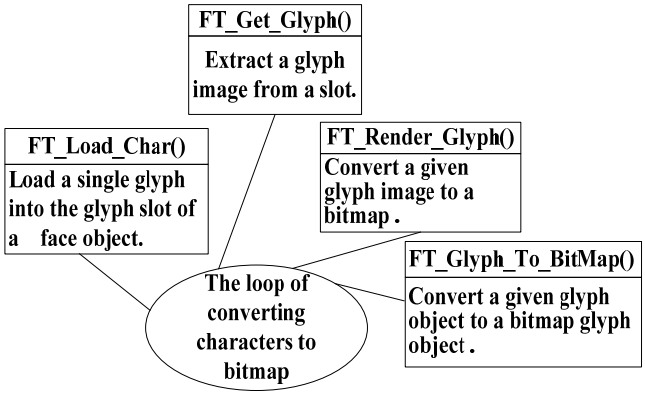

Figure 2.Approach of producing bitmap

2.2.4 SETTING GEOMETRICAL VERTEX AND TEXTURE DATA: It is necessary to get a handle of a glyph after producing a bitmap of the glyph image for us accessing the size of the bitmap and the horizontal distance from pen position to left borderline and the vertical distance from pen position to top borderline. We use these bitmaps as the textures of character models in OpenGL ES. The texture's size in OpenGL ES must be integral TH power of two. Converting bitmap's size to be true of the need of texture's size in OpenGL ES is necessary. It is realized by the follow methods:

```
int width = next_p2(bitmap.width);
int height = next_p2(bitmap.rows);
```

The function of next_p2 returns a value which is the approximation of integral TH power of two. There is a point on baseline named pen position. The pen position is used to give an orientation of glyph. The projection coordinate in screen of the given region's spherical coordinate is set as the label's initial pen position. The X-coordinate of every character glyph image's current pen position is the totting-up of all the anterior characters' advance widths basing on the initial pen position in the order of the character's appearing order in the character string. In the practical navigation application, the city position is express as latitude, longitude and altitude in spherical reference frame. The using of Cartesian reference frame in the process of projection in OpenGL ES makes it is necessary to transform the spherical coordinate to Cartesian coordinate. Some certain ground object's spherical coordinate is set as (lat,lon,alt) and its Cartesian coordinate is Q(X,Y,Z). X, Y and Z are calculated as follows:

$$X = alt \times \cos\left(\frac{lat \times \pi}{180}\right) \times \cos\left(\frac{lon \times \pi}{180}\right);$$

$$Y = alt \times \cos\left(\frac{lat \times \pi}{180}\right) \times \sin\left(\frac{lon \times \pi}{180}\right);$$

$$Z = alt \times \sin\left(\frac{lon \times \pi}{180}\right);$$

On the assumption that the camera's projection reference center is RCenter(rX,rY,rZ),the initial pen position is the projection coordinate what Q is relative to RCenter in screen.

Character geometrical model's width and height are the width and height of the character's glyph bitmap. Vertex X-coordinate at the bottom left corner of the model is what the X-coordinate of current pen position has been made a rightward translation whose movement distance is the horizontal distance from pen position on baseline to the leftmost pixel of glyph image along plus x-axis. Increasing Y-coordinate corresponding to downward scanning beam is realized by subtracting the distance from pen position to top. So the vertex Y-coordinate at the bottom left corner is the subtraction of screen height and the Y-coordinate of current pen position which has been made a downward translation whose movement distance is the vertical distance from pen position to the top of glyph image along plus y-axis. We accord the width and height to gain the rest vertices of the model.

3. RESULTS

Glyph measure adopts horizontal layout in the process of converting English characters to glyph bitmaps. Many characters' glyph bitmaps which are used as textures have the different width and height. The distances from pen position to the top borderline in most English characters glyph images are different from each other. Using the unchanged data from bitmap to set the Y-coordinate of character geometrical model's vertices finally displays irregular arranged character model which is shown in s60 simulator as Figure 3.

Figure 3. Irregular arranged English character models

It is necessary to adjustment the Y-coordinate of the character model for text alignment before displaying the text label. The final English labels display is shown in Figure 4.

Figure 4. Regular English character models

Glyph measure adopts vertical layout in the process of convert Chinese characters to glyph bitmaps. There is tiny difference between the distances from pen position to the top borderline of their bitmaps. Directly using the data from bitmap to set character geometrical model Y-coordinate has no influence in justification. The Chinese label is shown in Figure 5.

Figure 5. Chinese character models

When we zoom in or rotate the 3D scene, the character models' particulars and appearances keep the original shapes.

4. CONCLUSIONS

The Chinese characters and English characters are truly displayed in screen of cellphone by this method. We give the detailed approach in displaying text basing on Symbian OS and OpenGL ES. The detail of programming is related to the efficiency of program. It provides a convenient method to draw character model in 3D scene base on OpenGL or OpenGL ES in the paper.

Symbian OS has a great progress in developing. It provides a perfect support for OpenGL ES. More 3D games and 3D navigation systems can be used in cellphone in Symbian OS. More and more all-sided functions increase the more value and appeal for smart phone. More new ideas and new innovation

will be applied to Symbian OS and more chances and time will mature the strongpoint of cellphone.

REFERENCES

Edward Angel,2007. *Interactive Computer Graphics-A Top-Down Approach Using OpenGL*. Graphics program, pp.48-56

Dave Shreiner,Mason Woo,Jackie Neider,Tom Davis,2005. *OpenGL Programming Guide*.

Richard Harrison,2005. *Symbian OS C++ for Mobile Phones volume2*.

Ruiter H., Benhabib B.,(2008) Visual-model-based, real-time 3D pose tracking for autonomous navigation: methodology and experiments. In Auton Robot (2008) 25: 267–286.

Martı´n S., Sua´rez J., Orea R., Rubio R., Gallego R.(2009) GLSV: Graphics library stereo vision for OpenGL. In Virtual Reality (2009) 13: 51–57.