**第四章：图元详解（一）**

**文字详解**

这篇文章主要对图元中生成文字的方法进行详细讲解。

[再论文字]

在前几个章节中，我们主要讨论了图元的基本绘制方法，其中文字部分只是粗略地讲解了一下，那么本节将会进行进一步讲解。

文字，其本质意义上不属于图形（或者说属于比较复杂的、不易于抽象的图形），因此在Opengl中，若想生成文字，必须先生成文字图像，再将图像转化为纹理，绑定到图形上。

[几种文字的产生方式]

在geiv中，引擎为文字纹理的生成提供了一系列API，使这个过程大大简化，开发者可以使用下列方式产生需要的文字。

[方式1]给定文字序列的长宽，指定文字的字体、字号、内容、风格（指加粗、倾斜等）、颜色等属性，以及相对于给定区域的缩进位移，进而生成一块完整的纹理。

例子：

public static void main(String[] args) {

UESI UES = new R();

Obj font = UES.creatObj(UESI.BGIndex);

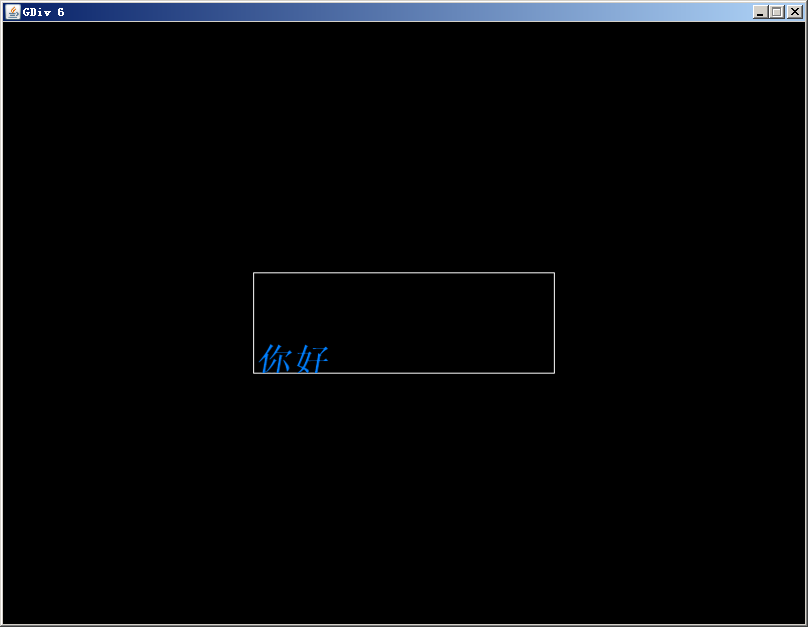
font.addGLRect("FFFFFF",0,0,300,100);//我们画一个矩形，标注这个300\*100的区域

font.addGLFont("007FFF",0,0,300,100,"宋体",Font.BOLD|Font.ITALIC,35,"你好",0,0);//之后绘制文字

font.setPosition(CANExPos.POS\_CENTER);

font.show();}

我们看一下这个结果：



效果不是很好对吧？有必要进行一些解释：

font.addGLFont("007FFF",0,0,300,100,"宋体",Font.BOLD|Font.ITALIC,35,"你好",0,0);

007FFF是颜色，

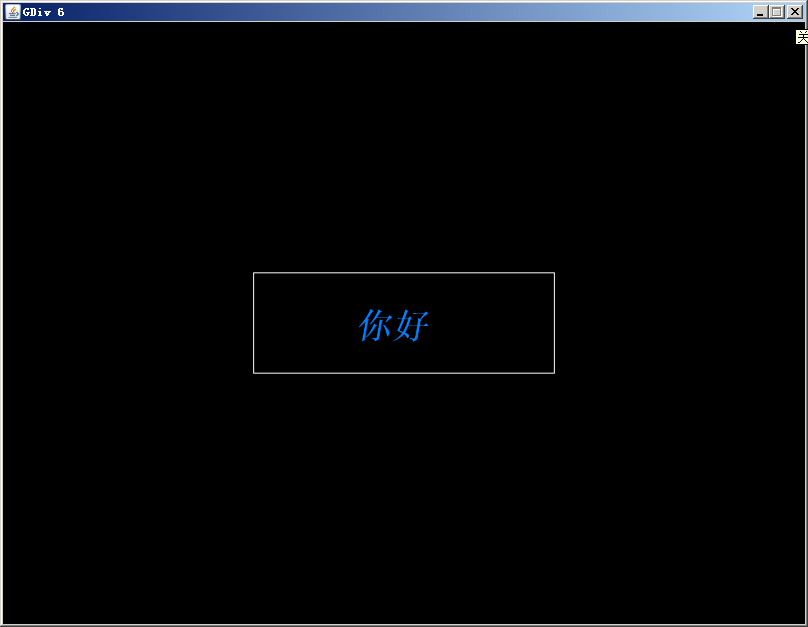
0,0,300,100,是绘制区域，同时是生成纹理的实际大小，可以理解为画纸大小

后面的”宋体”是字体，当然，如果使用英文字体是不能绘制出中文的，而且有些字体在Linux下是没有的，这点请注意

后面一个int变量用来指明字样式，可以是Font.PLAIN、Font.BOLD、Font.ITALIC，也可以像上例一样进行按位或操作取它们的叠加。

“你好”是文字的内容，不用多说。

后面的0,0是左缩进和下缩进，咱们的文字顶在左下角很不协调，所以需要调整缩进值，进行微移，将这里的0,0改为100,35后，结果如下：



但是这种绘制方式能满足的需求很有限，首先来讲，文字的画纸大小需要我们自己设定，由于字号与像素大小的关系比较难以把握，申请的区域一般比文字所占区域大，既浪费了空间，有很难单次绘制出满意的效果，需要做多次调整，这样会影响开发速度。

其次，输入的文字内容被传唤为一整个纹理，如果文字是一个易变变量，例如游戏分数，那么每次分数的改变，都需要生成一个纹理，这既不科学又严重影响效率。

我只建议这个方法用在已有图形上生成静态文字的情况下，除此之外使用它是相当不明智的。

[方式2]：使用内建字库

这种方案是由方案1演变而来。

在显示文字前，我们先创建一个字库，这个过程给定文字的各种属性，类似于1中的参数。

字库一旦创建，就具有全局效应，可以再其他类的其他地方使用。

例子：

public static void main(String[] args) {

UESI UES = new R();

Obj font = UES.creatObj(UESI.BGIndex);

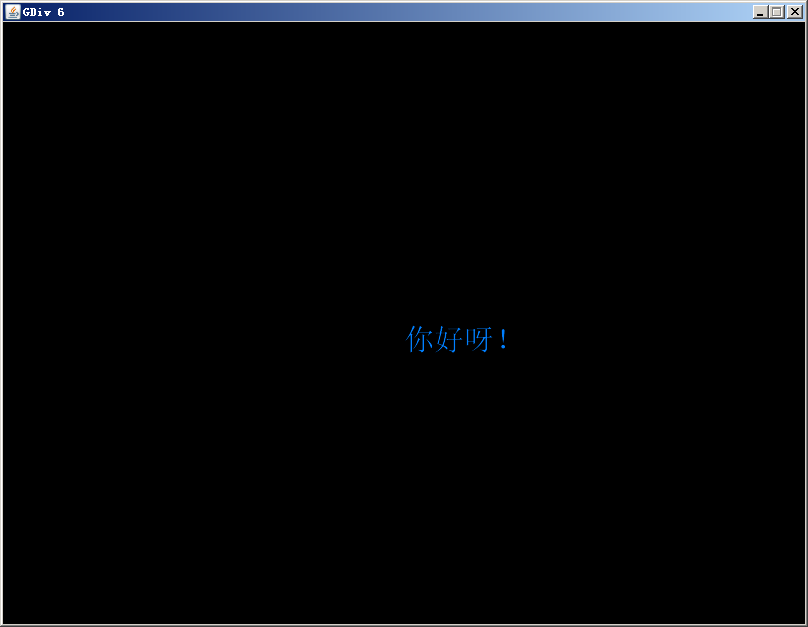
UES.addKWordTYPE("myKeyName","宋体","007FFF",30,Font.PLAIN,30,30,0,3);

font.addGLWordSet(0,0,"myKeyName","你好呀！");

font.setPosition(CANExPos.POS\_CENTER);

font.show();

}



UES.addKWordTYPE("myKeyName",

"宋体","007FFF",30,Font.PLAIN,30,30,0,3);

该语句生成了一个名字为”myKeyName”的字库，

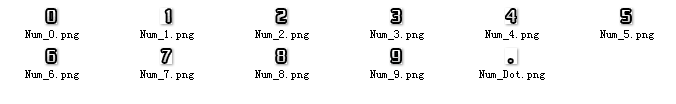
后面参数依次为：字体、颜色、字号、样式、每个字的大小、每个字的缩进值。

font.addGLWordSet(0,0,"myKeyName","你好呀！");

该语句使用名字为”myKeyName”的字库，分别生成”你”、”好”、”呀”、”！”四个字的纹理，当遇到相同的字时，会直接引用已经存在的纹理而不会重新生成，所以它比较适合经常变化的文字区域。

[方式3]：使用外建字库

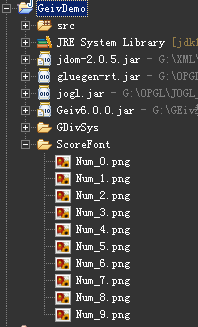
该方式由方式2演变而来，方式1、2的实现依赖于Java2D的文字API，生成的文字类型比较有限，例如像边框文字就无法绘制了，有很多时候，我们希望由外部的其他资源生成一个字库。

例如，我们有资源：

我们想使用这个数字生成一个数字字库该如何呢？

例子：

我们先将字库资源放到项目工作目录中，创建名为ScoreFont的文件夹:



public static void main(String[] args) {

UESI UES = new R();

Obj font = UES.creatObj(UESI.BGIndex);

UES.addKWordTYPE("myKeyName");

for(char c = '0'; c <= '9';c++){

UES.setKWord("myKeyName",c,".\\ScoreFont\\Num\_" + c + ".png");

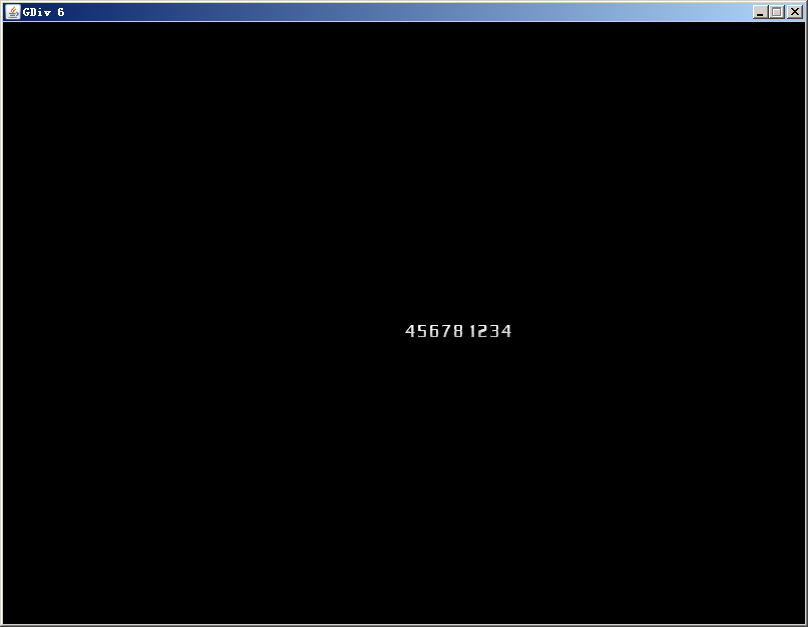
}

font.addGLWordSet(0,0,"myKeyName","456781234");

font.setPosition(CANExPos.POS\_CENTER);

font.show();

}

结果：

方式3与方式2不同，这里使用的：UES.addKWordTYPE("myKeyName");

并没有指明字体的产生细节，它仅仅创建了一个名字为myKeyName的空字库。

UES.setKWord("myKeyName",c,".\\ScoreFont\\Num\_" + c + ".png");

setKWord将指定字库的文字char c，与一个外部图形资源关联，资源可以是任何常见图片类型的路径，如jpg、png、bmp等。生成的字库纹理大小取决于图片的大小。

之后，字库的使用方式与方式2中没有差别，但是如果出现了字库中没有的字，则会产生异常，这点请注意。

[文字的更改]

无论使用以上三种方式的哪一钟，创建出的字体图形均满足CANFont接口，因此，统一在图元上使用setFontString(String S);与getFontString();即可设置、获取字体内容了，同样，它们具有图元索引重载。

**抛弃String** 在设置字体的过程中，如果使用了字库模式，则会为每个字保存生成过的纹理，这大大减少了内存的浪费，但是，如果您依然使用String类型作为字体处理的媒介的话，那么，String的碎片会生成的到处都是，以分数为例，纹理字库避免了重复纹理的生成，但是”0000”与”0001”依然是两个字符串，在用于分数或时间高速变化的情景中，资源浪费还是很高。

在图元里与字库相关的API中，添加了使用char数组的内容链接方式，对数组的改动可以反映在字库内容上，从而避免了String碎片的产生。

例子：还记得动态绘制章节的SerialTask吗？这次我们来生成动态增加的数字吧！

[源码打包]SRC：Sample\ Sample-Font\ DynamicNumber

Main.java

public class Main{

public static void main(String[] args) {

UESI UES = new R();

new ScoreFont(UES);

}

}

ScoreFont.java

public class ScoreFont implements SerialTask{

char[] score;

Obj font;

public ScoreFont(UESI UES){

font = UES.creatObj(UESI.BGIndex);

UES.addKWordTYPE("myKeyName");

for(char c = '0'; c <= '9';c++){

UES.setKWord("myKeyName",c,".\\ScoreFont\\Num\_" + c + ".png");

}

score = new char[6];

Arrays.fill(score, '0');

font.addGLWordSet(0,0,"myKeyName","000000");

**font.referanceKeyWord(score);//关联一个外部char数组。**

font.setPosition(CANExPos.POS\_CENTER);

font.show();

UES.addSerialTask(this);

}

@Override

public void Serial(int clock) {

addNum(5);

}

private void addNum(int index){

if(score[index] < '9'){

score[index] ++;

}else{

score[index] = '0';

addNum(index - 1);

}

}

}

上例生成了一个动态增加的数字，每秒增加60次。

…………

[总结]本节详细介绍了文字API的使用方法。

首先介绍了三种字体生成方式，分别是：

粗糙的统一生成方式：适用于静态字体。

内建字库方式：使用系统内部API绘制字体的字库，适合易变文字。

外建字库方式：使用系统外的资源生成字库，适用于内建字库不能满足需求的情况。

之后我们介绍了文字的获取与设定方法，也介绍了高频变换下放弃String的合理性与相应的解决方案。