

原16/24点阵字库（经典）

2012年02月15日 16:16:41 杨柳 阅读数：30403

7

7

比如汉字“中”的内码是0xd6d0,而他的区位码是5448.这个是怎么转换的呢？区位码不是内码减去0xa0a0吗？怎么算的呢？

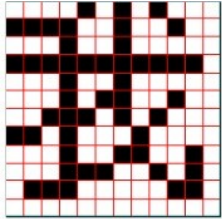
问题补充：

找到答案了：
汉字机内码、国标码和区位码三者之间的关系为：

- 区位码（十进制）的两个字节分别转换为十六进制后加20H得到对应的国标码；
 - 机内码是汉字交换码（国标码）两个字节的最高位分别加1，即汉字交换码（国标码）的两个字节分别加80H得到对应的机内码；
 - 区位码（十进制）的两个字节分别转换为十六进制后加A0H得到对应的机内码。
- 即：
- 区位码（十进制）—— +2020H（8224）——>国标码（十六进制）—— +8080H（32896）——>机内码（十六进制）

所有的汉字或者英文都是下面的原理，
由左至右，每8个点占用一个字节，最后不足8个字节的占用一个字节，而且从最高位向最低位排列。
生成的字库说明：（以12×12例子）

前5行：
1: 00001010 1000 → 0x0a 0x80
2: 11110010 0100 → 0xf2 0x40
3: 00010010 0000 → 0x12 0x00
4: 11111111 1110 → 0xff 0xe0
5: 00010010 0000 → 0x12 0x00



一个汉字占用字节数：12÷8=1·····4也就是占用了2×12=24个字节。
编码排序A0A0→A0FE A1A0→A2FE依次排列。
以12×12字库的“我”为例：“我”的编码为CED2，所以在汉字排在CEH-A0H=2EH区的D2H-A0H=32H个。所以在12×12字库的起始位置就是[{FE-A0}*2EH+32H]*24=104976开始的24个字节就是我的点阵模。
其他的类推即可。
英文点阵也是如此推理。

在DOS程序中使用点阵字库的方法



首先需要理解的是点阵字库是一个数据文件,在这个数据文件里面保存了所有文字的点阵数据.至于什么是点阵,我想我不讲大家都知道 的,使用类的电子辞典吧,那个的液晶显示器上面显示的汉子就能够明显的看出“点阵”的痕迹.在 PC 机上也是如此,文字也是由点阵来组成了,不同的是,PC机分辨率更高,高到了我们肉眼无法区分的地步,因此“点阵”的痕迹也就不那么明显了.

点阵、矩阵、位图这三个概念在本质上是有关联的,从某种程度上讲,这三个就是同义词.点阵从本质上讲就是单色位图,他使用一个比特来表示这个比特为0,表示某个位置没有点,如果为1表示某个位置有点.矩阵和位图有着密不可分的联系,矩阵其实是位图的数学抽象,是一个二维的阵列.位图的阵列,这个阵列中的 (x,y) 位置上的数据代表的就是对原始图形进行采样量化后的颜色值.但是,另一方面,我们要面对的问题是,计算机中数据的存放是线性的.因此,我们需要 将二维的数据线性化到一维里面去.通常的做法就是将二维数据按行顺序的存放,这样就线性化到了一维.

那么点阵字的数据存放细节到底是怎么样的呢.其实也十分的简单,举个例子最能说明问题.比如说 16*16 的点阵,也就是说每一行有16个点,由一个比特来表示,如果这个比特的值为1,则表示这个位置有点,如果这个比特的值为0,则表示这个位置没有点,那么一行也就需要16个比特,而8个字节就是,也就是说,这个点阵中,一行的数据需要两个字节来存放.第一行的前八个点的数据存放在点阵数据的第一个字节里面,第一行的后面八个点的数 据 存放在第二个字节里面,第二行的前八个点的数据存放在点阵数据的第三个字节里面,···,然后后面的就以此类推了.这样我们可以计算出存放一个点阵.....需要看下面这个图形化的例子:



```
| | |1|1| |1|1|1|1|1|1|1|1| | |
| | | |1| | | | | | |1| | | |
|1| | | | |1| | | | |1| | | |
| |1|1| | | |1| | | | |1| | | |
| | |1| | | |1| | | | |1| | | |
| | | |1| | |1| | | |1| | | | |
| | | |1| | | |1| | |1| | | | |
| | |1| | | | |1| |1| | | | | |
|1|1|1| | | | | |1| | | | | | |
| | |1| | | | | |1| |1| | | | |
| | |1| | | | |1| | | | | | | |
| | |1| | |1| | | | | |1|1|1| |
| | | |1| | | | | | | |1| | | |
| | | | | | | | | | | | | | | |
```

可以看出这是一个“汉”字的点阵,当然文本的方式效果不是很好. 根据上面的原则,我们可以写出这个点阵的点阵数据:0x40, 0x08, 0x37, 0xfc, 0x00, 当然写这个确实很麻烦所以我不再继续下去. 我这样做,也只是为了向你说明,在点阵字库中,每一个点阵的数据就是按照这种方式存放的.

当然也存在着不规则的点阵,这里说的不规则,指的是点阵的宽度不是8的倍数,比如 12*12 的点阵,那么这样的点阵数据又是如何存放的呢?其实一行的前面8个点存放在一个字节里面,每一行的剩下的4点就使用一个字节来存放,也就是说 剩下的4个点将占用一个字节的高4位,而这个字节的低4位;都默认为零. 这样做当然显得有点浪费,不过却能够便于我们进行存放和寻址. 对于 其他不规则的点阵,也是按照这个原则进行处理的. 这样我们可以得的点阵所占用的字节数为 (m+7)/8*n.

在明白了以上所讲的以后,我们可以写出一个显示一个任意大小的点阵字模的函数,这个函数的功能是输出一个宽度为w,高度为h的字模到屏幕的出,文字的颜色为 color,文字的点阵数据为 pdata 所指:

```
/*输出字模的函数*/

void _draw_model(char *pdata, int w, int h, int x, int y, int color)
{
    int i;          /* 控制行 */
    int j;          /* 控制一行中的8个点 */
    int k;          /* 一行中的第几个“8个点”了 */
    int nc;         /* 到点阵数据的第几个字节了 */
    int cols;       /* 控制列 */

    BYTE static mask[8]={128, 64, 32, 16, 8, 4, 2, 1}; /* 位屏蔽字 */

    w = (w + 7) / 8 * 8; /* 重新计算w */

    nc = 0;

    for (i=0; i<h; i++)
    {
        cols = 0;

        for (k=0; k<w/8; k++)
        {
```

7

7













```
for (j=0; j<8; j++)

{

if (pdata[nc]&mask[j])

putpixel(x+cols, y+i, color);

cols++;


}


nc++;


}


}


}
```


7


7











代码很简单,不用怎么讲解就能看懂,代码可能不是最优化的,但是应该是最易读懂的. 其中的 putpixel 函数,使用的是TC提供的 Graphics.h 头文件中的,用这个函数就可以完成点阵任意大小的点阵字模的输出.



接下来的问题就是如何在汉子库中寻址某个汉字的点阵数据了. 要解决这个问题,首先需要了解汉字在计算机中是如何表示的. 在计算机中英文可用 ASCII 码来表示,而汉字使用的是扩展 ASCII 码,并且使用两个扩展 ASCII 码来表示一个汉字. 一个 ASCII 码使用一个字节表示,所谓扩展 ASCII 码,也就是高位是1的 ASCII 码,简单的说就是码值大于等于 128 的 ASCII 码. 一个汉字由两个扩展 ASCII 码组成,第一个扩展 ASCII 码用来存放区码,第二个扩展 ASCII 码用来存放位码. 在 GB2312-80 标准中, 将所有的汉字分为94个区, 每个区有94个位可以存放94个汉字, 形成了人们常说的区位码, 这样总共就有 94*94=8836个汉字. 在点阵字库中, 汉字点阵数据就是按照这个区位的顺序来存放的, 也就是最先存放的是第一个区的汉字点阵数据, 在每一个区中有是按照位的顺序来存放的. 汉字区位码的存放实在扩展 ASCII 基础上存放的, 并且将区码和位码都加上了32, 然后存放在两个扩展 ASCII 码中. 具体的说就是:

第一个扩展ASCII码 = 128+32 + 汉字区码

第二个扩展ASCII码 = 128+32 + 汉字位码

如果用char hz[2]来表示一个汉字, 那么我可以计算出这个汉字的区位码为:

区码 = hz[0] - 128 - 32 = hz[0] - 160

位码 = hz[1] - 128 - 32 = hz[1] - 160.

这样, 我们可以根据区位码在文件中进行寻址了, 寻址公式如下:

汉字点阵数据在字库文件中的偏移 = ((区码-1) * 94 + 位码) * 一个点阵字模占用的字节数

在寻址以后, 即可读取汉字的点阵数据到缓冲区进行显示了. 以下是实现代码:

```
/* 输出一个汉字的函数 */

void _draw_hz(char hz[2], FILE *fp, int x, int y, int w, int h, int color)

{

char fontbuf[128]; /* 足够大的缓冲区,也可以动态分配 */

int ch0 = (BYTE)hz[0]-0xA0; /* 区码 */

int ch1 = (BYTE)hz[1]-0xA0; /* 位码 */

/* 计算偏移 */

long offset = (long)pf->_hz_buf_size * ((ch0 - 1) * 94 + ch1 - 1);

fseek(fp, offset, SEEK_SET); /* 进行寻址 */

fread(fontbuf, 1, (w + 7) / 8 * h, fp); /* 读入点阵数据 */

_draw_model(fontbuf, w, h, x, y, color); /* 绘制字模 */

}
```

以上介绍完了中文点阵字库的原理, 当然还有英文点阵字库了. 英文点阵字库中单个点阵字模数据的存放方式与中文是一模一样的, 也就是对我们 _draw_model 函数同样可以使用到英文字库中. 唯一不同的是对点阵字库的寻址上. 英文使用的就是 ASCII 码, 其码值是0到127, 寻址公式为:

英文点阵数据在英文点阵字库中的偏移 = 英文的ASCII码 * 一个英文字模占用的字节数

可以看到,区分中英文的关键就是,一个字符是 ASCII 码还是扩展 ASCII 码,如果是 ASCII 码,其范围是0到127,这样是使用的英文字母,如果是扩展 ASCII 码,则与其后的另一个扩展 ASCII 码组成汉字内码,使用中文字库进行显示.只要正确区分 ASCII 码的类型并进行分别的处理,也就能实现中英文混排了.

点阵字库和矢量字库的差别



我们都只知道,各种字符在**电脑**屏幕上都是以一些点来表示的,因此也叫点阵.最早的字库就是直接把这些点存储起来,就是点阵字库.常见的点阵字库有 16x16, 24x24 等.点阵字库也有很多种,主要区别在于其中存储编码的方式不同.点阵字库的最大缺点就是它是固定分辨率的,也就是每种字库只能按固定的尺寸下使用,效果很好,但如果将其放大或缩小使用,效果就很糟糕了,就会出现我们通常说的锯齿现象.因为需要的字体大小组合有无数种,所以不可能都定义一个点阵字库.于是就出现了矢量字库.

点阵字库

点阵字库是把每个字符的笔划分解成各种直线和曲线,然后记下这些直线和曲线的参数,在显示的时候,再根据具体的尺寸大小,画出这些点阵,就得到了该字符.它的好处就是可以随意放大缩小而不失真.而且所需存储量和字符大小无关.矢量字库有很多种,区别在于他们采用的不同数学模型来描述字符.常见的矢量字库有 Type1字库和Truetype字库.

在点阵字库中,每个字符由一个位图表示(如图2.5所示),并把它用一个称为字符掩膜的矩阵来表示,其中的每个元素都是一位二进制数,如果该位的笔画经过此位,该像素置为字符颜色;如果该位为0,表示字符的笔画不经过此位,该像素置为背景颜色.点阵字符的显示分为两步:首先从字库中将它的位图检索出来,然后将检索到的位图写到帧**缓冲器**中.

在实际应用中,同一个字符有多种字体(如宋体、楷体等),每种字体又有多种大小型号,因此字库的存储空间十分庞大.为了减少存储空间,一般采用

矢量字符记录字符的笔画信息而不是整个位图,具有存储空间小,美观、变换方便等优点.例如:在AutoCAD中使用图形实体-形(Shape)-来定义矢量字符.它采用了直线和圆弧作为基本的笔画来对矢量字符进行描述.对于字符的旋转、放大、缩小等几何变换,点阵字符需要对其位图中的每个象素进行变换,而矢量字符只需要对其几何图素进行变换就可以了,例如:对直线笔画的两个端点进行变换,对圆弧的起点、终点、半径和圆心进行变换等等.

矢量字符的显示也分为两步.首先从字库中将它的字符信息,然后取出端点坐标,对其进行适当的几何变换,再根据各端点的标志显示出字符.

轮廓字形法是当今国际上最流行的一种字符表示方法,其压缩比大,且能保证字符质量.轮廓字形法采用直线、B样条/Bezier曲线的集合来描述一线.轮廓线构成一个或若干个封闭的平面区域.轮廓线定义加上一些指示横宽、竖宽、基点、基线等等控制信息就构成了字符的压缩数据.

如何使用Windows的系统字库生成点阵字库？



我的程序现在只能预览一个汉字的**不同字体**的点阵表达.

界面很简单: 一个输出点阵大小的选择列表(8x8, 16x16, 24x24等),一个系统中已有的字体名称列表,一个预览按钮,一块画图显示区域.

得到字体列表的方法:(作者称这一段是用来取回系统的字体,然后添加到下拉框中)

```
//取字体名称列表的回调函数,使用前要声明一下该方法
int CALLBACK MyEnumFONtProc (ENUMLOGFONTEX* lpelf, NEWTEXTMETRICEX* lpntm, DWORD nFontType, long lParam)
{
    CFontPeekerDlg* pWnd=(CFontPeekerDlg*) lParam;

    if (pWnd)
    {
        if ( pWnd->m_combo_sfont.FindString(0, lpelf->elfLogFont.lfFaceName) < 0 )
        {
            pWnd->m_combo_sfont.AddString(lpelf->elfLogFont.lfFaceName);
        }

        return 1;
    }

    return 0;
}

//说明:CFontPeekerDlg 是我的dialog的类名, m_combo_sfont是列表名称下拉combobox关联的control变量
```

//调用的地方 (****问题1:下面那个&lf怎么得到呢……)

```
{
    ::EnumFontFamiliesEx((HDC)      dc,&lf,      (FONTENUMPROC)MyEnumFontProc,(LPARAM)      this,0);

    m_combo_sfont.SetCurSel(0);
}
```

字体预览:

如果点阵大小选择16,显示的时候就画出16x16个方格.自定义一个类CMyStatic继承自CStatic,用来画图.在CMyStatic的OnPaint()函数

取得字体:

常用的方法:用CreateFont创建字体,把字TextOut再用GetPixel()取点存入数组. 缺点:必须把字TextOut出来,能在屏幕上看见,不

我的方法,用这个函数:GetGlyphOutline(),可以得到一个字的轮廓矢量或者位图.可以不用textout到屏幕,直接取得字模信息

函数原型如下:

```
DWORD      GetGlyphOutline(
    HDC      hdc,                      //画图设备句柄
    UINT      uChar,                  //将要读取的字符/汉字
    UINT      uFormat,                //返回数据的格式(字的外形轮廓还是字的位图)
    LPGLYPHMETRICS    lpgm,          //    GLYPHMETRICS结构地址,输出参数
    DWORD      cbBuffer,              //输出数据缓冲区的大小
    LPVOID      lpvBuffer,            //输出数据缓冲区的地址
    CONST      MAT2      *lpmat2      //转置矩阵的地址
);
```

说明:

uChar字符需要判断是否是汉字还是英文字符.中文占2个字节长度.

lpgm是输出函数,调用GetGlyphOutline()是无须给lpgm 赋值.

lpmat2如果不需要转置,将 eM11.value=1; eM22.value=1; 即可.

cbBuffer缓冲区的大小,可以先通过调用GetGlyphOutline(……lpgm, 0, NULL, mat); 来取得,然后动态分配lpvBuffer,再一
GetGlyphOutline,将信息存到lpvBuffer. 使用完后再释放lpvBuffer.

程序示例:(**问题2:用这段程序,我获取的字符点阵总都是一样的,不管什么字……)

……前面部分省略……

```
GLYPHMETRICS    glyph;

MAT2      m2;

memset(&m2, 0, sizeof(MAT2));

m2.eM11.value    =    1;

m2.eM22.value    =    1;

//取得buffer的大小

DWORD      cbBuf    =    dc.GetGlyphOutline(    nChar,    GGO_BITMAP,    &glyph,    0L,    NULL,    &m2);

BYTE*      pBuf=NULL;

//返回GDI_ERROR表示失败.

if(    cbBuf    !=    GDI_ERROR    )
```

```
{
    pBuf = new BYTE[cbBuf];

    //输出位图GGO_BITMAP 的信息. 输出信息4字节 (DWORD) 对齐

    dc.GetGlyphOutline( nChar, GGO_BITMAP, &glyph, cbBuf, pBuf, &m2);

}

else
{
    if(m_pFont!=NULL)

    delete m_pFont;

    return;

}

}
```

编程中遇到问题:

一开始,GetGlyphOutline总是返回-1,getLastError显示是“无法完成的功能”,后来发现是因为调用之前没有给hdc设置Font.

后来能取得pBuf信息后,又开始郁闷,因为不太明白bitmap的结果是按什么排列的. 后来跟踪汉字“一”来调试(这个字简单),注意到了glyph. gmBlackBoxX就是输出位图的宽度, glyph. gmBlackBoxY就是高度. 如果gmBlackBoxX=15, glyph. gmBlackBoxY=2, 表示输出的pBuf中有这些信息: 位图有2行信息, 每行15 bit来存储信息.

例如:我读取“一”:glyph. gmBlackBoxX = 0x0e, glyph. gmBlackBoxY=0x2; pBuf长度cbBuf=8 字节

pBuf信息: 00 08 00 00 ff fc 00 00

字符宽度 0x0e=14 则 第一行信息为: 0000 0000 0000 100 (只取到前14)

第二行根据4字节对齐的规则, 从0xff开始 1111 1111 1111 110

看出“一”字了吗?呵呵

直到他的存储之后就可以动手解析输出的信息了.

我定义了一个宏#define BIT(n) (1<<(n)) 用来比较每一个位信息时使用

后来又遇到了一个问题,就是小头和大头的问题了. 在我的机器上是little endian的形式, 如果我用

```
unsigned long *lptr = (unsigned long*)pBuf;
```

```
//j from 0 to 15
```

```
if( *lptr & BIT(j) )
```

```
{
```

```
//这时候如果想用j来表示写1的位数, 就错了
```

```
}
```

因为从字节数组中转化成unsigned long型的时候, 数值已经经过转化了, 像上例中, 实际上是0x0800 在同BIT(j) 比较.

不多说了, 比较之前转化一下就可以了if(htonl(*lptr) & BIT(j))

Unicode中文点阵字库的生成与使用



点阵字库包含两部分信息. 首先是点阵字库文件头信息, 它包含点阵字库文字的字号、多少位表示一个像素, 英文字母与符号的size、起始和结束码、在文件中的起始偏移, 汉字的size、起始和结束unicode编码、在文件中的起始偏移. 然后是真实的点阵数据, 即一段段二进制串, 每一串表示一个汉字的点阵信息.

要生成点阵字库必须有文字图形的来源, 我的方法是使用ttf字体. ttf字体的显示采用的是SDL_ttf库, 这是开源图形库SDL的一个扩展库, 用libfreetype以读取和绘制ttf字体.

它提供了一个函数,通过传入一个Unicode编码便能输出相应的文字的带有alpha通道的位图。那么我们可以扫描这个位图以得到相应文字的点阵(alpha通道,我们可以在点阵信息中也加入权值,使得点阵字库也有反走样效果。我采用两位来表示一个点,这样会有三级灰度(还有一个表示透明)。

点阵字库的显示首先需要将文件头信息读取出来,然后根据unicode编码判断在哪个区间内,然后用unicode编码减去此区间的起始unicode,再加上此区间的文件起始偏移得到文件的绝对偏移,然后读出相应位数的数据,最后通过扫描这段二进制串,在屏幕的相应位置输出点阵字型。

显示点阵字体需要频繁读取文件,因此最好做一个固定大小的缓存,采用LRU置换算法维护此缓存,以减少磁盘读取。

标准点阵字库芯片



最近重新找了一下C语言的资料，深深的被c语言的底层操作特性迷恋~。在这方面，最经典的一本书莫过于清华大学出版社的《C高级实用程序设计》（王士元），这是见过的写的最好的一本书，非常可惜的这本书现在已经绝版了（可能是因为技术发展和更新的太快），在书店里网上都无法买到了。记得本科时期经常不释手，里面的知识极具魅力，即使今天看起来仍让我觉得不是过时，而是回味无穷。这里提到的字库文件和操作系统都已经属于古董级别了，现在可能也很难找到。现在时代也很少有人研究了，但我想在单片机等嵌入式系统的点阵式汉字显示屏中仍然在使用。

在这里我参考了一些资料中的用C语言显示汉字字库的资料。在计算机发展的早期，为了支持显示汉字，国内发明了相应的2个字节表示的汉字国标（GB）码，分为94个区，每区94个汉字，汉字在其所在区内的位置用位号表示，两个字节分别表示区号和位号，为了区分ASCII码，每个字节的首位都被置为1。在网络传输时，通常不细述这些细节了。在国际编码中，中国汉字被分配到第16区（起始区号0x0F）。为了显示汉字，需要汉字字形文件（字库）的支持。在DOS时代，出现了USC-16（16*16字形），HZK16（16*16字形），HZK24（24*24字形）等字库文件，本质上一个汉字字符是一个二值图像（即bpp=1），所以本质上是一种图形字体（非矢量的），例如每个像素，每个像素占1位（1/8byte），因此每个汉字在文件中占据了 $16*16/8=32$ bytes/汉字。HZK24每个汉字占据 $24*24/8=72$ bytes/汉字。由于采用了这些约定，每个汉字附加描述信息，而全部是紧密排列的像素字形点阵，文件从第一个字节开始第一个字符一直到最后一个字符结束，文件也没有后缀名。字形在文件中是紧密排列的，但一个重要问题是：

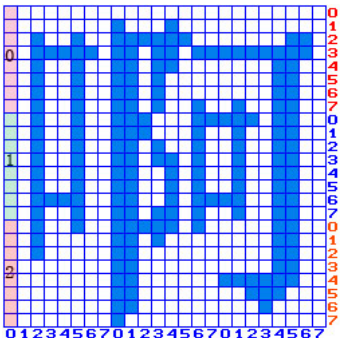
(1) 字形扫描顺序:

HZK16是按行扫描，而HZK24是按列扫描。

所以假设一个字形读取到一个byte[]。则字节的分布分为按行扫描和按列扫描。例如在HZK16中是按行一行一行扫描，即前byte[0],byte[1]是第一行，接着2个byte是第2行，依此类推。而在HZK24中是按列扫描，因为打印机有一个纵向的24针，每一次可以逐列的打印一行汉字，因此为了配合打印机，HZK24采用了按列扫描。（图中的数字表示的该字节在byte[]中的索引。）

0	1	0	3		69
2	3	1	4		70
•	•	2	5		71
30	31				

下面这张图显示的是24*24像素的按列扫描的汉字字符，是如何用一个BYTE[]数组来描述的：图中用彩色填充的方式标示出了BYTE[0],BYTE[1],BYTE[2]：



(2) 字形在字库文件中的定位:

请注意的，在HZK16和HZK24中定位是不同的。这是由于汉字在文件中的起始区不同。HZK16的汉字起始区在第16个分区，而HZK24则直接从第一个分区开始。HZK16的分区15里面存储了一些特殊符号，字母等（见后面的截图）。因此汉字所在的分区是从15号分区开始的，而HZK24不包含前面的特殊字符，第一个区就是汉字区“啊”，从0号分区开始。需要注意的是，两者的【bytes/汉字】数值不同。用offset表示从文件头开始计算的文件偏移地址，code[2]表示汉子码。

则code是用内码表示的，如汉字表中的第一个汉字“啊”的内码是{0xb0,0xa1}；从汉字码的第一个字节获取区号，从第二个字节获取位号。将他们减去0xA1就转换为和位号。即换算成区号是{0x0f,0x00}，表示“啊”字位于第15区，位号是0。

对于HZK16来说：94是每个区的汉字数。

```
unsigned long offset=( (code[0]-0xA1) *94 + (code[1]-0xA1) ) *32L; //32是每个汉字占据的字节数
```

对于HZK24来说:

```
unsigned long offset=( (code[0]-0xA1-15) *94 + (code[1]-0xA1) ) *72L;//72是每个汉字占据的字节数
```

注意上面的HZK24中，由于第一个分区就是汉字区，所以区号被减去了0x0F。

这样，我们看显示一个字符的代码：（以下代码来自于《C高级应用程序设计》一书。）

```

<!--HZK16显示汉字的代码
Code highlighting produced by Actipro CodeHighlighter (freeware)
http://www.CodeHighlighter.com/

--> /*在屏幕的x0, y0位置显示一个汉字字符*/
void dishz(int x0,int y0,char code[],int color)
{
    unsigned char mask[]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
    /*屏蔽字模每行各位的数组*/
    int i,j,x,y,pos;

```



```
char mat[32];
/*下面这个函数用于在文件中读取相应的字节到mat数组中*/
get_hz(code,mat);
y=y0;
for (i=0;i<16;++i)
{
x=x0;
pos=2*i;
for(j=0;j<16;++j)
{
/*屏蔽出汉字字模的一个位，即一个点是1 则显示，否则不显示该点*/
if ((mask[j%8]&mat[pos+j/8])!=NULL)
putpixel(x,y,color);
++x;
}
++y;
}
```

👍7

💬7

📖

🔖

📱

<

>

我们要注意的，由于每个汉字占用了2个字节，因此我们必须每次是字符串的指针递增2。例如：

```
char* s="这是汉字字符串";
while(*s)
{
dishz(x0,y0,s,LIGHTBLUE);
x0+=16;
s+=2; //notice: not s++
}
```



上面的字模数组的每个数字的某一位为1，这样用位与操作可以判断某个byte中的某一位为1还是0，如果为1表示这里应该显示汉字的前景颜色。上面的代码是用于HZK1描的顺序显示的。按列显示的代码原理类似，不再列出。

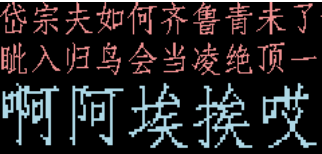
关于加载字模，主要采用上面的文件定位，读取一定数量字节即可，这里我们也不再列出。

下面的截图是HZK16字库的前面一部分特殊字符（有象棋棋子，制表符，全角的字母数字等等）：（每行32个字符，每个字符的左上角被我标记了一个红点）。

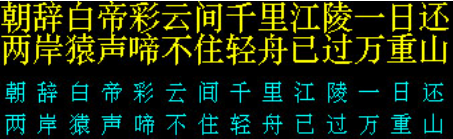


上面的截图采用了在《用c语言显示BMP》一文中的TC截屏代码截屏的代码（我将截屏代码写到cpyscr.h文件中放到了include文件夹下，这样只要用#include "cpyscr.h"截屏函数了），注意第一行中的数字序号表示的是该字符的序号（注意该序号不是ASCII码！！），而不是文件地址！。

下图是用HZK24F（仿宋体）显示的汉字，上面是原样大小输出（24*24），较大的字是把字形边长扩大成2倍边长（48*48）的输出，由于是位图字体，所以扩大后会有位图感：



宋体汉字库：



注意国标码，区位码和内码的关系和转换关系

🔖 收藏

🔗 分享

陈小春坦言：这游戏不充钱都能当全服大哥，找到充值入口算我输！

贪玩游戏 · 顶新



想对作者说点什么

下载16X16点阵字库

03-05

windows 自带的汉字点阵字库，省得费时费力寻找了。版权归微软所有，免资源积分

安装和激活Office 2019

有条件请支持正版！相比费尽力气找一个可能不太安全的激活工具，直接买随时随地更新的Office 365确实是最好...

来自： [过了即是客](#)

如何把unicode和点阵字库结合起来

最近改写了大学时候写的一些代码，是关于点阵字库的，能把给DOS的程序用在MP4上，到是我原来没想到的。由...

来自： [vrix的专栏](#)

下载

16X16,8X8点阵字库

07-07

这里有很多字库这里有很多字库这里有很多字库这里有很多字库这里有很多字库这里有很多字库

下载

常用的ASCII显示8x8点阵字库

03-01

一些常用的8x8点阵字体 可以直接在程序中调用这些点阵数组来实现字体的显示 以前做LCD显示的时候找了很久，这是在UC_GUI的压缩包中抽取出来的，希望对大家有用。

一插上电,50平米内都暖和了!3天一度电,今日特惠!

优诺·顶新

下载

HZK12,HZK16,HZK24,HZK32,HZK40,HZK48,ASC12, ASC16,ASC24, ASC32, ASC48字库+读取...

02-27

这几天接了一个项目，需要制作点阵文字，所以接触了字库这东西，由于网上字库挺多的，但是不全，有的字库全，但是字库的读取代码只有一部分，所以还是花费了一天时间搞这...

下载

中文点阵字体

06-30

POS机字体，各种小票字体，迎合所需，禁止使用到不正当领域。

下载

最新unicode点阵字库生成工具（ fontmaker ）

04-25

v1.06 (20120410) FontMaker(点阵字库) 1. 增加系统字体支持，操作更简单，快捷。 v1.05 (20120324) FontMaker(点阵字库) 1. 修正了字符对齐问题

下载

8x8点阵字库测试软件

05-21

8x8LED点阵屏的字库字模生成器测试器

Python实现点阵字体读取与转换

点阵字体是指根据文字的像素点来显示的字体，效果如下： 使用Python读取并显示的过程如下： 根据中文字符获...

来自： [像风一样的自由](#)

别再玩假传奇了，这款传奇爆率9.8，你找到充值入口算我输！

贪玩游戏·顶新

下载

最全HZK12,HZK16,HZK24,HZK32,HZK40,HZK48,ASC12, ASC16,ASC24, ASC32, ASC48字库+...

01-13

这几天接了一个项目，需要制作点阵文字，所以接触了字库这东西，由于网上字库挺多的，但是不全，有的字库全，但是字库的读取代码只有一部分，所以还是花费了一天时间搞这...

gbk点阵字库，16x16大小

文件拓展名为fon，系统是打不开的，可以使用WinHex观察存储信息 可用于SD卡中字库的存储

下载

汉子、ASCII码点阵字库 12、16、24、48点阵字库

10-24

包含常用所有点阵字库，12、16、24、48

下载

最全的点阵字库（ ASC12, ASC16, HZK16, HZK24,... ）

09-13

最全的点阵字库，包括ASC12, ASC16, HZK12, HZK14, HZK16, HZK24, HZK24F, HZK24H, HZK24K, HZK24S, HZK24T等。

下载

多国语言开发必备工具(点阵字库+多国语言文本管理+MTK 字库+字库解析源码 c)

08-21

v1.11 (08/14/2012) 1. 增加注册年限选择。 v1.10 (06/26/2012) 1. 精简字库检索表。 2. 增加 GBK 字库输出。 3. 增加一种点阵数据存储格式（只存有效像素数据--not fixed, 不支...

别再玩假传奇了，这款传奇爆率9.8，你找到充值入口算我输！

贪玩游戏·顶新

下载

8*16点阵字库ASC16

07-29

UCDOS的8*16点阵字库文件，用于LCD显示英文、数字和符号

下载

GB2312 16*16点阵字库

05-22

GB2312 16*16点阵字库

<div>24x24点阵字库的显示</div> <div>首先，研究了很久，以为和16x16没多大区别，后来又发现了很多24x24需要注意的点，如不要小看这个按列扫描， ...</div> <div>来自：HYDMonster的博客</div>	551	
<div>24X24 黑体简体中文点阵字库</div> <div>使用矢量字库在屏幕尺寸较小的设备上面，显示相对模糊，效果不如点阵字库。另外，矢量字库engine会消耗更好...</div> <div>来自：demonjeremy的专栏</div>	2471	
<div><div>下载</div>24点阵字库</div> <div>24点阵字库用于打印。把rar解压缩后，使用ts24.lib即可。</div>	11-16	
<div>上班族的选择！起初不想玩，现在玩上瘾了，限本月本地用户！</div> <div>热门推荐</div>		
<div>51单片机 16X16点阵循环滚动显示汉字（基于proteus）</div> <div>原创文章，转载注明地址作者，作者TEER。 一、效果展示 二、所用材料 1.16X16点阵 Proteus并不提供16X16的...</div> <div>来自：qq_41639829的博客</div>	1481	
<div>16X16汉字点阵显示</div> <div>#include #define SIZE 16 using namespace std; int main() { int screen[SIZE][SIZE]; un...</div> <div>来自：AWEN的专栏</div>	563	
<div><div>下载</div>16x16点阵汉字显示</div> <div>用点阵写自己要显示的汉字，容易掌握！就是代码有点难写！希望能帮到大家</div>	12-13	
<div>【MSP430】16 * 16 汉字点阵</div> <div>/*****EC实验室*****/ *****/ ...</div> <div>来自：qq_21989461的博客</div>	662	
<div><div>下载</div>点阵字库12*12等</div> <div>12*12，16*16等点阵字库比较全面</div>	01-30	
<div><div>下载</div>点阵字库生成源码(linux)</div> <div>把矢量字库转为点阵字库。 1.在linux环境下编译运行。 2.使用了freetype。 3.可指定点阵的宽高。</div>	01-21	
<div>UCDOS中的点阵字库HZK12,HZK16,HZK24,ASC12,ASC16(转)</div> <div>原文：http://cache.baidu.com/c?m=9f65cb4a8c8507ed4fece7631046893b4c4380147780914c34c3933fc23...</div> <div>来自：jszj的专栏</div>	8554	
<div><div>下载</div>16*16点阵汉字库</div> <div>这是一个很好用的16*16点阵汉字库，压缩包里面有详细的使用说明。 我在这里简要说明一下点阵汉字显示的原理：点阵汉字的显示其实是在屏幕上画点，以16*16点阵汉字为例， ...</div>	12-12	
<div>ttf字库</div> <div>FreeType 2 教程第二步 管理字型@ 2009 David Turner (david@freetype.org)@ 2009 The FreeType Develop...</div> <div>来自：sup_xiaok的专栏</div>	916	
<div><div>下载</div>GBK汉字16*16和24*24点阵字模，用于显示和打印，包含索引算法及工具</div> <div>个人整理的专业的GBK汉字点阵字库字模，包含GBK1.0中22046个汉字，不含用户自定义区的汉字。包含完整的汉字内码扩展规范(GBK)编码表两份，一个是按分区一分按编码顺序...</div>	08-12	
<div><div>下载</div>12*12,24*24点阵的字库，并有字库生成程序，随心所欲生成各种字库</div> <div>12*12,24*24点阵的字库，并有字库生成程序，随心所欲生成各种字库</div>	04-22	
<div><div>下载</div>点阵字库文件-ASC16-24-HZK16-24-32-40-48</div> <div>点阵字库 ASC16 ASC24 HZK16 HZK24 HZK32 HZK40 HZK48 库文件，用法demo-C语言源码，运行截图及说明</div>	02-06	
<div><div>下载</div>点阵字库GB2312</div> <div>点阵字库 GB2312 16*16 24*24 宋体 宋体加粗</div>	07-18	
<div>点阵字库HZK12 HZK16 HZK24 ASC12 ASC16 简介 及 使用方法[附源码]</div> <div>如何在嵌入式系统中使用大量的汉字和字符呢？ DOS前辈们经过艰辛的努力，将制作好的字模放到了一个个标...</div> <div>来自：u011766068的专栏</div>	4848	
<div><div>下载</div>汉字库点阵24*24HZK24</div> <div>HZK24+这个文件内含有中文标点符号，本人亲测，在计算偏移地址时不需要往前偏移15个区，偏移地址计算公式如下： offset = (94*qh+wh)*72;/*得到偏移位置*/ 使用时注意， ...</div>	10-23	



最新文章

- 关于phpstorm+xdebug调试失败记录
- 已到中年，但不得不继续疯狂
- 网络工具netcat
- Qt5.8 for Android配置
- centos6.x搭建SVN服务器

个人分类

linux系统服务	20篇
qt技术	15篇
web后台技术	19篇
web前端技术	7篇
c/c++技术	11篇

展开

归档

2019年1月	1篇
2018年6月	1篇
2017年3月	6篇
2016年11月	16篇
2016年10月	3篇

展开

热门文章

- linux动态编译和静态编译
阅读量：30920
- 16/24点阵字库（经典）
阅读量：30401
- setsockopt的作用
阅读量：21810
- QT历届版本下载总汇
阅读量：21015
- ffmpeg命令使用详解
阅读量：17785

最新评论

- 关于sip应用于公网上的一些尴尬
sea2air：可以联系你吗？有SIP项目请教。可以加Q241383661
- git还原某个特定的文件到之前的版本
qungmu：[code=plain] git checkout [<tree>] [->] <...>
- 关于sip应用于公网上的一些尴尬

7

7



elvaqing23：博主现在还在从事呼叫相关的工作吗?现在已经是行业的大牛了吧，膜拜。

已到中年，但不得不继续疯狂
weixin_35971458：想联系你下，谢谢！

已到中年，但不得不继续疯狂
changyinling520：学无至今



联系我们



微信客服





QQ客服

🔔 QQ客服 ✉ kefu@csdn.net
🗣 客服论坛 ☎ 400-660-0108
 🕒 工作时间 8:30-22:00

关于我们 招聘 广告服务 网站地图
🔍 百度提供站内搜索 京ICP证09002463号
©1999-2019 江苏乐知网络技术有限公司
江苏知之为计算机有限公司 北京创新乐知
信息技术有限公司版权所有

网络110报警服务 经营性网站备案信息
北京互联网违法和不良信息举报中心
中国互联网举报中心


7


7

