步骤一、依据《字库生成操作方法.docx》生成需要的字库。

步骤二、如果是裸机工程，需要将resource\lvgl添加到工程中；

如果是包含LVGL的工程，那么只需要单独添加  
 《fakeFs.c》《fakeFs.h》《lv\_font\_map\_loader.c》《lv\_font\_map\_loader.h》  
 这四个文件即可。

步骤三、使用VS2019将步骤一的文件转化为bin文件。步骤如下：

#define FONT\_C2BIN 1

#if (FONT\_C2BIN == 1)

#include <stdio.h>

#include <string.h>

typedef struct

{

    char name[32];   /\* 资源的名字 \*/

    uint32\_t cnt;    /\* 词条数量 \*/

    uint32\_t add;    /\* 地址 \*/

    uint32\_t size;   /\* 资源的大小 \*/

    uint32\_t offset; /\* 资源相对文件偏移 \*/

} Catalog\_Small\_Typedef;

//生成文件

//long int len :文件长度

//void \*pSor :写入文件数据

//char \*Name :生成文件名

//返回： 0-成功 -1失败

int CreatFile(FILE\* fp, uint32\_t len, void\* pSor, char\* Name, uint32\_t cnt)

{

    uint32\_t lenTrue = len;

    if (len % 16 != 0)

    {

        lenTrue += 16 - len % 16;

    }

    uint32\_t tell1 = ftell(fp);

    Catalog\_Small\_Typedef Catalog;

    Catalog.add = (uint32\_t)pSor;

    Catalog.size = lenTrue;

    Catalog.cnt = cnt;

    Catalog.offset = ftell(fp) + sizeof(Catalog\_Small\_Typedef);

    memset(Catalog.name, 0, sizeof(Catalog.name));

    memcpy(Catalog.name, Name, strlen(Name));

    fwrite(&Catalog, 1, sizeof(Catalog\_Small\_Typedef), fp);

    if (len % 4 == 0)

    {

        fwrite(pSor, 4, len / 4, fp);

    }

    else

    {

        fwrite(pSor, 4, len / 4, fp);

        fwrite(pSor, 1, len % 4, fp);

    }

    uint8\_t tmp = 0;

    if (lenTrue > len)

    {

        while (len < lenTrue)

        {

            fwrite(&tmp, 1, 1, fp);

            len++;

        }

    }

    return 0;

}

int main(void)

{

    int status;

    char\* FileName = "font16.FON";

    FILE\* fp;

    status = fopen\_s(&fp, FileName, "wb+");

    if (fp == NULL) {

        printf("create:%s fail ! \r\n", FileName);

        return -1;

    }

    Catalog\_Small\_Typedef Catalog;

    Catalog.add = 0;

    Catalog.size = sizeof(Catalog\_Small\_Typedef);

    Catalog.cnt = 15;

    Catalog.offset = ftell(fp) + sizeof(Catalog\_Small\_Typedef);

    memset(Catalog.name, 0, sizeof(Catalog.name));

    memcpy(Catalog.name, FileName, strlen(FileName));

    fwrite(&Catalog, 1, sizeof(Catalog\_Small\_Typedef), fp);

    /\* 01 \*/ status = CreatFile(fp, sizeof(SiYuan16\_bpp2), (void\*)&SiYuan16\_bpp2, "lv\_font\_t", 1);

    /\* 02 \*/ status = CreatFile(fp, sizeof(font\_dsc), (void\*)&font\_dsc, "font\_dsc", 1);

    /\* 03 \*/ status = CreatFile(fp, sizeof(kern\_pairs), (void\*)&kern\_pairs, "kern\_pairs", 1);

    /\* 04 \*/ status = CreatFile(fp, sizeof(cmaps), (void\*)&cmaps, "cmaps", font\_dsc.cmap\_num);

    /\* 05 \*/ status = CreatFile(fp, sizeof(unicode\_list\_2), (void\*)&unicode\_list\_2, "unicode\_list\_2", 1);

    /\* 06 \*/ status = CreatFile(fp, sizeof(unicode\_list\_6), (void\*)&unicode\_list\_6, "unicode\_list\_6", 1);

    /\* 07 \*/ status = CreatFile(fp, sizeof(unicode\_list\_8), (void\*)&unicode\_list\_8, "unicode\_list\_8", 1);

    /\* 08 \*/ status = CreatFile(fp, sizeof(unicode\_list\_10), (void\*)&unicode\_list\_10, "unicode\_list\_10", 1);

    /\* 09 \*/ status = CreatFile(fp, sizeof(unicode\_list\_14), (void\*)&unicode\_list\_14, "unicode\_list\_14", 1);

    /\* 10 \*/ status = CreatFile(fp, sizeof(unicode\_list\_17), (void\*)&unicode\_list\_17, "unicode\_list\_17", 1);

    /\* 11 \*/ status = CreatFile(fp, sizeof(unicode\_list\_20), (void\*)&unicode\_list\_20, "unicode\_list\_20", 1);

    /\* 12 \*/ status = CreatFile(fp, sizeof(glyph\_bitmap), (void\*)&glyph\_bitmap, "glyph\_bitmap", 1);

    /\* 13 \*/ status = CreatFile(fp, sizeof(glyph\_dsc), (void\*)&glyph\_dsc, "glyph\_dsc", sizeof(glyph\_dsc) / sizeof(lv\_font\_fmt\_txt\_glyph\_dsc\_t));

    /\* 14 \*/ status = CreatFile(fp, sizeof(kern\_pair\_values), (void\*)&kern\_pair\_values, "kern\_pair\_values", 1);

    /\* 15 \*/ status = CreatFile(fp, sizeof(kern\_pair\_glyph\_ids), (void\*)&kern\_pair\_glyph\_ids, "kern\_pair\_glyph\_ids", 1);

     fclose(fp);

     if (status == 0) {

         printf("Finish ! \r\n");

     }

     return 0;

 }

 #endif

具体说明：  
①结构体typedef struct

{

char name[32]; /\* 资源的名字 \*/

uint32\_t cnt; /\* 词条数量 \*/

uint32\_t add; /\* 地址 \*/

uint32\_t size; /\* 资源的大小 \*/

uint32\_t offset; /\* 资源相对文件偏移 \*/

} Catalog\_Small\_Typedef;

的内容需要与lv\_font\_map\_loader.h的  
typedef struct{ char name[32]; /\* 资源的名字 \*/ uint32\_t cnt; /\* 词条数量 \*/ uint32\_t add; /\* 资源假地址 \*/ uint32\_t size; /\* 资源的大小 \*/ uint32\_t offset; /\* 资源相对文字的地址 \*/} Catalog\_Font\_Typedef;

结构上完全一致，不然无法解析字库资源。

② unicode\_list\_xxx部分是根据不同字符大小会有所变动，动态调整。

步骤四、将生成好的FON文件使用《2字体文件打包\_V2.exe》进行打包。

该打包工程的索引格式为

/\* 字库目录信息类型 \*/

typedef struct

{

    char name[36];   /\* 资源的名字 \*/

    uint32\_t check;   /\* 资源的和校验 \*/

    uint32\_t size;   /\* 资源的大小 \*/

    uint32\_t offset; /\* 资源相对于基地址的偏移 \*/

} CatalogTypeDef\_fakeFs;

与fakefs需要保持完全一致。

步骤五、

将打包好的文件烧录到指定地址中去。

步骤六、  
按照如下方式加载字体库。

fakeFs\_file\_t file;

        fakeFs\_init(0x60800000,(4 \* 1024));

        res = fakeFs\_open(&file, "font16.FON");

        if(!res)

        {

            font16 = lv\_load\_map\_font(&file, (uint32\_t)file.file\_add);

            fakeFs\_close(&file);

        }

        res = fakeFs\_open(&file, "font24.FON");

        if(!res)

        {

            font24 = lv\_load\_map\_font(&file, (uint32\_t)file.file\_add);

            fakeFs\_close(&file);

        }

fakeFs\_init中分别填写 资源的目录地址 和 资源的目录大小  
如果该文件是字体文件时，进一步调用lv\_load\_map\_font去加载字体。

步骤七、

运行lvgl库的工程关于该字体库如何使用无需再单独配置，但是裸机程序还需要再做额外配置。

如下：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*解析显示LVGL不等宽字库\*begin\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const uint8\_t \_lv\_bpp1\_opa\_table[2]  = {0, 255};          /\*Opacity mapping with bpp = 1 (Just for compatibility)\*/

const uint8\_t \_lv\_bpp2\_opa\_table[4]  = {0, 85, 170, 255}; /\*Opacity mapping with bpp = 2\*/

const uint8\_t \_lv\_bpp3\_opa\_table[8]  = {0, 36,  73, 109,   /\*Opacity mapping with bpp = 3\*/

                                        146, 182,  219, 255

                                       };

const uint8\_t \_lv\_bpp4\_opa\_table[16] = {0,  17, 34,  51,  /\*Opacity mapping with bpp = 4\*/

                                        68, 85, 102, 119,

                                        136, 153, 170, 187,

                                        204, 221, 238, 255

                                       };

const uint8\_t \_lv\_bpp8\_opa\_table[256] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,

                                         16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,

                                         32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,

                                         48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,

                                         64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79,

                                         80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95,

                                         96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111,

                                         112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127,

                                         128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,

                                         144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159,

                                         160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175,

                                         176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191,

                                         192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,

                                         208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223,

                                         224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239,

                                         240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255

                                        };

#define LV\_IS\_ASCII(value)              ((value & 0x80U) == 0x00U)

#define LV\_IS\_2BYTES\_UTF8\_CODE(value)   ((value & 0xE0U) == 0xC0U)

#define LV\_IS\_3BYTES\_UTF8\_CODE(value)   ((value & 0xF0U) == 0xE0U)

#define LV\_IS\_4BYTES\_UTF8\_CODE(value)   ((value & 0xF8U) == 0xF0U)

#define LV\_IS\_INVALID\_UTF8\_CODE(value)  ((value & 0xC0U) != 0x80U)

const lv\_font\_t \* font16 = NULL;

const lv\_font\_t \* font24 = NULL;

/\*\*

 \* Decode an UTF-8 character from a string.

 \* @param txt pointer to '\0' terminated string

 \* @param i start byte index in 'txt' where to start.

 \*          After call it will point to the next UTF-8 char in 'txt'.

 \*          NULL to use txt[0] as index

 \* @return the decoded Unicode character or 0 on invalid UTF-8 code

 \*/

uint32\_t lv\_txt\_utf8\_next(const char \* txt, uint32\_t \* i)

{

    /\*\*

     \* Unicode to UTF-8

     \* 00000000 00000000 00000000 0xxxxxxx -> 0xxxxxxx

     \* 00000000 00000000 00000yyy yyxxxxxx -> 110yyyyy 10xxxxxx

     \* 00000000 00000000 zzzzyyyy yyxxxxxx -> 1110zzzz 10yyyyyy 10xxxxxx

     \* 00000000 000wwwzz zzzzyyyy yyxxxxxx -> 11110www 10zzzzzz 10yyyyyy 10xxxxxx

     \*/

    uint32\_t result = 0;

    /\*Dummy 'i' pointer is required\*/

    uint32\_t i\_tmp = 0;

    if(i == NULL) i = &i\_tmp;

    /\*Normal ASCII\*/

    if(LV\_IS\_ASCII(txt[\*i])) {

        result = txt[\*i];

        (\*i)++;

    }

    /\*Real UTF-8 decode\*/

    else {

        /\*2 bytes UTF-8 code\*/

        if(LV\_IS\_2BYTES\_UTF8\_CODE(txt[\*i])) {

            result = (uint32\_t)(txt[\*i] & 0x1F) << 6;

            (\*i)++;

            if(LV\_IS\_INVALID\_UTF8\_CODE(txt[\*i])) return 0;

            result += (txt[\*i] & 0x3F);

            (\*i)++;

        }

        /\*3 bytes UTF-8 code\*/

        else if(LV\_IS\_3BYTES\_UTF8\_CODE(txt[\*i])) {

            result = (uint32\_t)(txt[\*i] & 0x0F) << 12;

            (\*i)++;

            if(LV\_IS\_INVALID\_UTF8\_CODE(txt[\*i])) return 0;

            result += (uint32\_t)(txt[\*i] & 0x3F) << 6;

            (\*i)++;

            if(LV\_IS\_INVALID\_UTF8\_CODE(txt[\*i])) return 0;

            result += (txt[\*i] & 0x3F);

            (\*i)++;

        }

        /\*4 bytes UTF-8 code\*/

        else if(LV\_IS\_4BYTES\_UTF8\_CODE(txt[\*i])) {

            result = (uint32\_t)(txt[\*i] & 0x07) << 18;

            (\*i)++;

            if(LV\_IS\_INVALID\_UTF8\_CODE(txt[\*i])) return 0;

            result += (uint32\_t)(txt[\*i] & 0x3F) << 12;

            (\*i)++;

            if(LV\_IS\_INVALID\_UTF8\_CODE(txt[\*i])) return 0;

            result += (uint32\_t)(txt[\*i] & 0x3F) << 6;

            (\*i)++;

            if(LV\_IS\_INVALID\_UTF8\_CODE(txt[\*i])) return 0;

            result += txt[\*i] & 0x3F;

            (\*i)++;

        }

        else {

            (\*i)++; /\*Not UTF-8 char. Go the next.\*/

        }

    }

    return result;

}

/\*\*

 \* @brief 获取字体的宽度

 \* @param {lv\_font\_t \*} font

 \* @param {uint32\_t} letter

 \* @return {\*}

 \* @note

 \*/

lv\_font\_glyph\_dsc\_t lv\_get\_fontSize\_letter(uint32\_t letter)

{

    const lv\_font\_t \* font = NULL;

    lv\_font\_glyph\_dsc\_t g;

    uint16\_t fontHeight = LCD\_GetFontCombine(FONT\_LVGL)->Height;

    if(fontHeight == 16)

    {

        font = font16;

    }

    else if(fontHeight == 24)

    {

        font = font24;

    }

    if(letter == ' ')

    {

        g.bpp = 1;

        g.adv\_w = 0;

        g.box\_h = fontHeight;

        g.box\_w = fontHeight/2;

        g.ofs\_x = 0;

        g.ofs\_y = 0;

        return g;

    }

    if(font == NULL)

    {

        memset(&g, 0, sizeof(lv\_font\_glyph\_dsc\_t));

        return g;

    }

    bool g\_ret = lv\_font\_get\_glyph\_dsc(font, &g, letter, '\0');

    if(g\_ret == false) {

        /\*Add warning if the dsc is not found

         \*but do not print warning for non printable ASCII chars (e.g. '\n')\*/

        if(letter >= 0x20 &&

           letter != 0xf8ff && /\*LV\_SYMBOL\_DUMMY\*/

           letter != 0x200c) { /\*ZERO WIDTH NON-JOINER\*/

           memset(&g, 0, sizeof(lv\_font\_glyph\_dsc\_t));

            // LV\_LOG\_WARN("lv\_draw\_letter: glyph dsc. not found for U+%" PRIX32, letter);

        }

    }

    return g;

}

void lv\_draw\_letter\_normal(const lv\_point\_t \*pos, lv\_font\_glyph\_dsc\_t \*g, const uint8\_t \*map\_p, uint8\_t enFillBackColor)

{

    const uint8\_t \* bpp\_opa\_table\_p;

    uint32\_t bitmask\_init;

    uint32\_t bitmask;

    uint32\_t bpp = g->bpp;

    lv\_opa\_t opa = LV\_OPA\_COVER;

    uint32\_t shades;

    pixel\_t colorB;

    pixel\_t colorT = LCD\_GetTextColor();

    int8\_t fontWidth = g->box\_w;

    uint16\_t fontHeight = LCD\_GetFontCombine(FONT\_LVGL)->Height;

    if(fontWidth > 32)

    {

        return;

    }

    if(bpp == 3) bpp = 4;

    switch(bpp) {

        case 1:

            bpp\_opa\_table\_p = \_lv\_bpp1\_opa\_table;

            bitmask\_init  = 0x80;

            shades = 2;

            break;

        case 2:

            bpp\_opa\_table\_p = \_lv\_bpp2\_opa\_table;

            bitmask\_init  = 0xC0;

            shades = 4;

            break;

        case 4:

            bpp\_opa\_table\_p = \_lv\_bpp4\_opa\_table;

            bitmask\_init  = 0xF0;

            shades = 16;

            break;

        case 8:

            bpp\_opa\_table\_p = \_lv\_bpp8\_opa\_table;

            bitmask\_init  = 0xFF;

            shades = 256;

            break;       /\*No opa table, pixel value will be used directly\*/

        default:

            LV\_LOG\_WARN("lv\_draw\_letter: invalid bpp");

            return; /\*Invalid bpp. Can't render the letter\*/

    }

    static lv\_opa\_t opa\_table[256];

    static lv\_opa\_t prev\_opa = LV\_OPA\_TRANSP;

    static uint32\_t prev\_bpp = 0;

    if(opa < LV\_OPA\_MAX) {

        if(prev\_opa != opa || prev\_bpp != bpp) {

            uint32\_t i;

            for(i = 0; i < shades; i++) {

                opa\_table[i] = bpp\_opa\_table\_p[i] == LV\_OPA\_COVER ? opa : ((bpp\_opa\_table\_p[i] \* opa) >> 8);

            }

        }

        bpp\_opa\_table\_p = opa\_table;

        prev\_opa = opa;

        prev\_bpp = bpp;

    }

    int32\_t col, row;

//    int32\_t box\_w = g->box\_w;

//    int32\_t box\_h = g->box\_h;

//    int32\_t width\_bit = box\_w \* bpp; /\*Letter width in bits\*/

    /\*Calculate the col/row start/end on the map\*/

    int32\_t col\_start = (fontWidth - g->box\_w) / 2 + pos->x;

    int32\_t col\_end   = col\_start + g->box\_w;

//    int32\_t col\_start = 0;

//    int32\_t col\_end   = g->box\_w;

    int32\_t row\_start = pos->y;

    int32\_t row\_end   = pos->y + g->box\_h;

    /\*Move on the map too\*/

//    uint32\_t bit\_ofs = (row\_start \* width\_bit) + (col\_start \* bpp);

    uint32\_t bit\_ofs = 0;

    map\_p += bit\_ofs >> 3;

    uint8\_t letter\_px;

    uint32\_t col\_bit;

    col\_bit = bit\_ofs & 0x7; /\*"& 0x7" equals to "% 8" just faster\*/

    uint32\_t col\_bit\_max = 8 - bpp;

//    uint32\_t col\_bit\_row\_ofs = (box\_w + col\_start - col\_end) \* bpp;

    uint32\_t col\_bit\_row\_ofs = 0;

//    // colorT，alphaT：表示前景色和前景色的透明度

//    // colorB，alphaB：表示背景色和背景色的透明度

//    // colorF，alphaF：表示计算得到的颜色和透明度

//    alphaF = alphaT + alphaB\*(1 - alphaT);

//    // r,g,b 分别计算

//    colorF = (colorT\*alphaT + colorB\*alphaB\*(1 - alphaT)) / alphaF;

    lv\_opa\_t alphaT;

    lv\_opa\_t alphaF;

    uint8\_t cr;

    uint8\_t cg;

    uint8\_t cb;

    pixel\_t colorF;

    uint32\_t tmp;

    if(enFillBackColor)

    {

        for(row = row\_start; row < row\_end; row++) {

            for(col = col\_start; col < col\_end; col++) {

                LCD\_DrawPixel((uint16\_t)(col), (uint16\_t)(row), (pixel\_t)LCD\_GetBackColor());

            }

        }

    }

    for(row = row\_start; row < row\_end; row++) {

        bitmask = bitmask\_init >> col\_bit;

        for(col = col\_start; col < col\_end; col++) {

            /\*Load the pixel's opacity into the mask\*/

            letter\_px = (\*map\_p & bitmask) >> (col\_bit\_max - col\_bit);

            if(letter\_px) {

                alphaT = bpp\_opa\_table\_p[letter\_px];

            }

            else {

                alphaT = 0;

            }

            /\*Go to the next column\*/

            if(col\_bit < col\_bit\_max) {

                col\_bit += bpp;

                bitmask = bitmask >> bpp;

            }

            else {

                col\_bit = 0;

                bitmask = bitmask\_init;

                map\_p++;

            }

            if(alphaT != 0)

            {

                if(alphaT == 255)

                {

                    LCD\_DrawPixel((uint16\_t)(col), (uint16\_t)(row), (pixel\_t)colorT);

                    // destBuf[row \* fontWidth + col] = colorT;

                }

                else

                {

                    if(fontHeight == 24 || (fontHeight == 16 /\* && alphaT < 128 \*/))

                    {

                        colorB = LCD\_GetPixel((uint16\_t)(col), (uint16\_t)(row));

                        // colorB = colorT;

                        alphaF = 255;

                        tmp = RGB\_R(colorT) \* alphaT / 255 + RGB\_R(colorB) \* (255 - alphaT) / alphaF;

                        cr = (uint8\_t)tmp;

                        tmp = RGB\_G(colorT) \* alphaT / 255 + RGB\_G(colorB) \* (255 - alphaT) / alphaF;

                        cg = (uint8\_t)tmp;

                        tmp = RGB\_B(colorT) \* alphaT / 255 + RGB\_B(colorB) \* (255 - alphaT) / alphaF;

                        cb = (uint8\_t)tmp;

                        colorF = RGB(cr, cg, cb);

                            LCD\_DrawPixel((uint16\_t)(col), (uint16\_t)(row), (pixel\_t)colorF);

                        // destBuf[row \* fontWidth + col] = colorF;

                    }

                }

            }

            else

            {

                // destBuf[row \* fontWidth + col] = colorB;

            }

        }

        col\_bit += col\_bit\_row\_ofs;

        map\_p += (col\_bit >> 3);

        col\_bit = col\_bit & 0x7;

    }

}

/\*\*

 \* @brief  显示一个LVGL字符

 \* @param  usX ：在特定扫描方向下字符的起始X坐标

 \* @param  usY ：在特定扫描方向下字符的起始Y坐标

 \* @param  usChar ：要显示的英文字符（国标码）

 \* @retval 无

 \*/

void LCD\_DispChar\_lvgl\_uniwidth(uint16\_t Xpos, uint16\_t Ypos, uint32\_t letter, uint8\_t enFillBackColor)

{

    const lv\_font\_t \* font = NULL;

    lv\_font\_glyph\_dsc\_t g;

    uint16\_t fontHeight = LCD\_GetFontCombine(FONT\_LVGL)->Height;

    uint8\_t baseLine;

    if(fontHeight == 16)

    {

        font = font16;

    }

    else if(fontHeight == 24)

    {

        font = font24;

    }

    else

    {

        return;

    }

    bool g\_ret = lv\_font\_get\_glyph\_dsc(font, &g, letter, '\0');

    if(g\_ret == false) {

        /\*Add warning if the dsc is not found

         \*but do not print warning for non printable ASCII chars (e.g. '\n')\*/

        if(letter >= 0x20 &&

           letter != 0xf8ff && /\*LV\_SYMBOL\_DUMMY\*/

           letter != 0x200c) { /\*ZERO WIDTH NON-JOINER\*/

            LV\_LOG\_WARN("lv\_draw\_letter: glyph dsc. not found for U+%" PRIX32, letter);

        }

        return;

    }

    /\*Don't draw anything if the character is empty. E.g. space\*/

    if((g.box\_h == 0) || (g.box\_w == 0) /\* || (g.box\_h > fontHeight) \*/)

        return;

    /\* 计算baseLine的位置(不根据LVGL格式进行进行) \*/

    if(fontHeight == 16)

    {

        baseLine = 2;

    }

    else

    {

        baseLine = 3;

    }

    lv\_point\_t gpos;

    gpos.x = g.ofs\_x / 2;

//    if((font->line\_height - font->base\_line > g.box\_h))

//    {

//        gpos.y = (font->line\_height - font->base\_line) - g.box\_h;

//    }

//    else

//    {

//        gpos.y = (font->line\_height - g.box\_h);

//    }

    /\* 确定纵向从第几行开始显示(不根据LVGL格式进行进行)\*/

    // gpos.y = (font->line\_height - font->base\_line) - g.box\_h - g.ofs\_y;

    // if(gpos.y + g.box\_h > fontHeight)

    // {

    //  gpos.y = 0;

    // }

    gpos.y = ((fontHeight - baseLine) - g.box\_h - g.ofs\_y);

    // if((fontHeight - baseLine > g.box\_h))

    // {

    //     gpos.y = ((fontHeight - baseLine) - g.box\_h - g.ofs\_y);

    // }

    // else

    // {

    //     gpos.y = (fontHeight - g.box\_h);

    // }

    gpos.x += (lv\_coord\_t)Xpos;

    gpos.y += (lv\_coord\_t)Ypos;

    //gpos.y = (font->line\_height - font->base\_line) - g.box\_h - g.ofs\_y;

    const uint8\_t \* map\_p = lv\_font\_get\_glyph\_bitmap(g.resolved\_font, letter);

    if(map\_p == NULL) {

        // LV\_LOG\_WARN("lv\_draw\_letter: character's bitmap not found");

        return;

    }

    if(g.resolved\_font->subpx) {

        // LV\_LOG\_WARN("Can't draw sub-pixel rendered letter because LV\_USE\_FONT\_SUBPX == 0 in lv\_conf.h");

    }

    else {

        lv\_draw\_letter\_normal(&gpos, &g, map\_p, enFillBackColor);

    }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*解析显示LVGL不等宽字库\*end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/