DJYOS
图形编程
（下）

目录

DJYOS 图形内核（GK） 用户手册

编写：贺敏 2015 年 02 月 12 日
Review：_年_月_日
审阅:罗侍田

1. 概述

DJYGUI 是运行在 DJYOS 的一个图形系统，包括 GK（图形内核）、GKLIB（GK 函数库）和 GDD（图形装饰平台）三大组件。GK（GUI KERNEL，即图形内核）是 DJYGUI 多窗口支持的底层核心，大部分的图形操作如剪切域、多窗口层叠、基本显示等都是在这里完成，它不是一堆显示函数的集合，而是一套比较完善的图形支持系统。

GK 提供图形操作的大部分功能，如多窗口管理、多显示器、镜像显示、色彩管理、窗口 ROP 属性和多种缓冲方式等，同时支持基本的图形绘制，如点、直线、圆、曲线、矩形填充和位图等。因此，GK 和 GKLIB 相结合，形成了一个精悍而强大的图形支持系统，特别适合于资源紧张的嵌入式系统，能够为应用程序提供完成中等复杂程度的图形编程。如果系统资源比较丰富，要实现复杂的图形功能，建议再把 GDD 模块加上。

在 DJYGUI 系统中，总体架构及层次关系如图 1-1 所示。

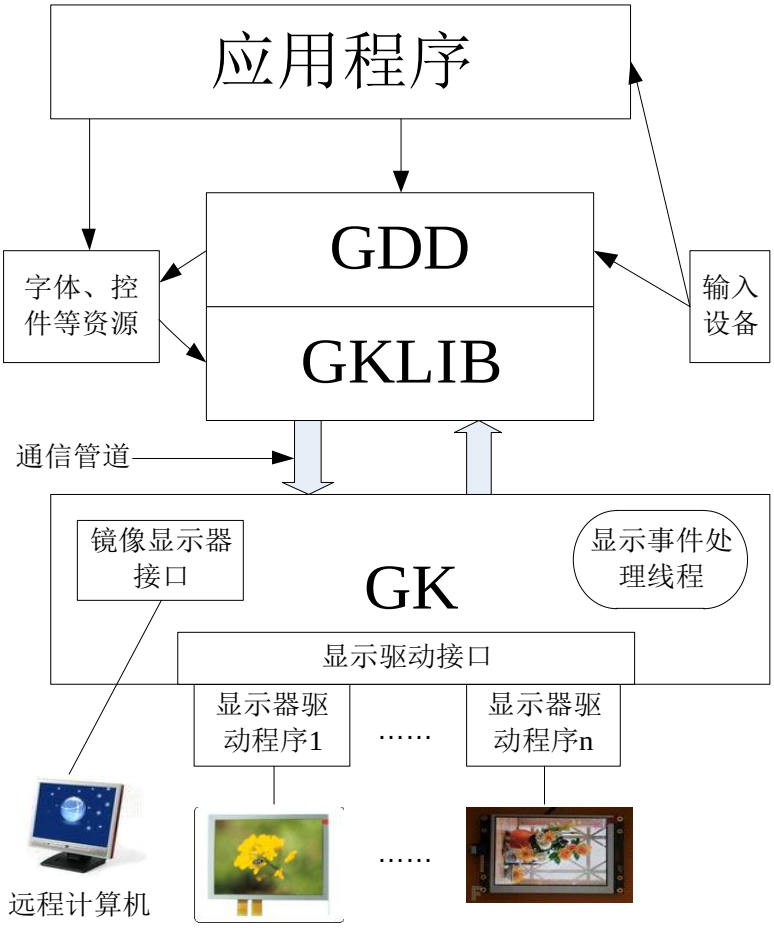

图 1-1 DJYGUI 总体架构

2. 窗口管理

2.1. 概述

GK 所指的窗口是一块矩形显示区域，它没有装饰、控件、输入焦点、菜单等窗口常见组件和属性，它是复杂窗口的原始雏形。因此，GK 中所述的窗口与 GDD 中的窗口应该区别对待，GDD 中的窗口是在原始窗口为基础上，增加了各种装饰、控件、输入焦点的窗口，由 GDD 管理。说明：本文档中所述的窗口，如无特殊说明，一概是指 GK 管理的原始窗口。

2.2. Z 序

在 GK 中，窗口在桌面上的排列顺序是按 Z 序方式排列。窗口的 Z 序，指明了窗口在重叠窗口堆中的位置，窗口堆沿着“ Z”轴方向，从屏幕上垂直向屏幕外延伸。Z 序顶部的窗口覆盖 Z 序底部的窗口。

在 DJYGUI 中，窗口是按资源的形式组织。在系统资源树中，有一个名为“ gkwindow”的根节点，该节点是所有窗口的共同祖先，由 GK_ModuleInit 函数创建。每台显示器的桌面，是“ gkwindow ”资源的子节点。每个桌面窗口有一套 Z 序系统。

窗口在资源树中的组织和它在Z序中的位置关系如图 2-1 所示。

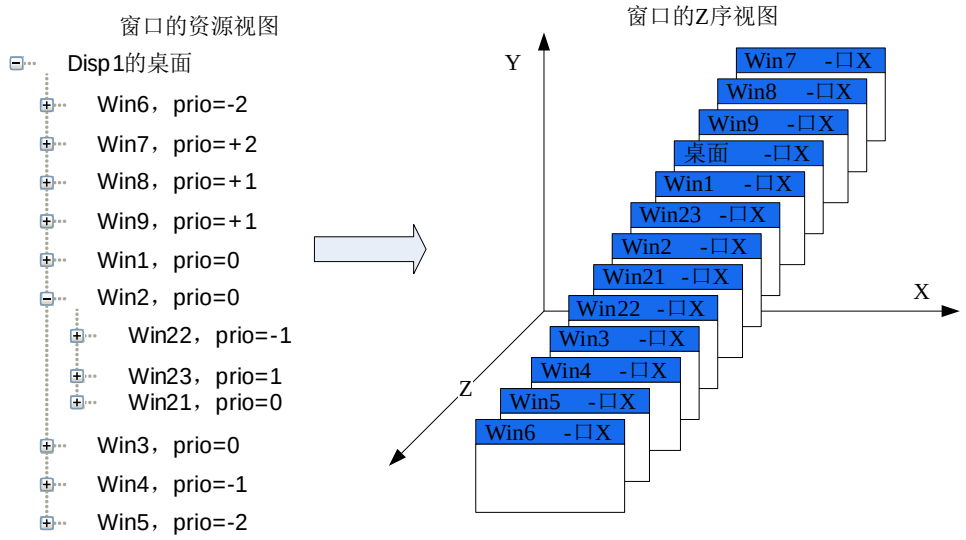

图 2-1 窗口组织和 Z 序

对窗口有关的资源结点的规则作如下说明：

1、同级窗口只使用前后结点，结点前一个为 previous，后一个为 next。

2、窗口存在父子关系，除桌面外其他窗口都有父窗口。

3、同级窗口中位于最前端的为父窗口指向的第一个窗口，其 next 指向同级的最后端窗口。

如图 2-1 所示，从窗口的资源视图中可得出，桌面包含多个子结点，这些子结点是桌面的子窗口，而子窗口也可包含子窗口，即桌面的孙窗口。窗口的 Z 序视图中，Win6 是窗口堆中 Z 序的最前端窗口，它会覆盖 Z 序中靠后的窗口。

注意，子窗口有可能覆盖父窗口，也有可能被父窗口覆盖。例如，Win6 是桌面的子窗口，覆盖桌面，而窗口 win7~9 是桌面的子窗口，但却被桌面遮盖。DJYGUI 的这种特性，可用于特殊的显示效果，如利用子窗口来做背景或利用子窗口来实现异形窗口等。

2.3. 优先级

窗口的优先级决定窗口在 Z 序中的排列位置。

窗口的优先级可通过 GKLIB 提供的 API 设置，窗口优先级的设定规则如下：

1、 同辈分的窗口中，优先级数值越小，优先级越高，在 Z 序中越靠前。

2、 桌面的优先级为 0，不可改变。优先级≤0 的窗口，优先级高于父窗口，在Z序中排列在父窗口前面；优先级<0 的窗口，优先级低于父窗口，在Z序中排列在父窗口后面。例如，图 2-1 中win1 优先级为 0，覆盖桌面，win9 优先级为 1，被桌面覆盖。

3、 任何一个窗口，连同其子孙窗口在 Z 轴中占据连续的区间，比如 win22 的优先级数小于 win3，但其在 Z 轴中却被 win3 覆盖，因为其父窗口 win2 是被 win3 覆盖的。同样的道理，也可以解释 win23 能够优先于 win1。

2.4. ROP 属性

DJYGUI 支持的光栅操作，包括二元光栅操作、alpha、透明色。

二元光栅操作是指在使用 GK 在窗口画点、线和填充区域时，GK 使用二元光栅操作码 ROP2 组合画笔或画刷像素和目标像素混合以得到新的目标像素。目前，GK 支持 16 种二元光栅操作，与 Windows 二元光栅操作兼容。二元光栅详细介绍见 3.3。

窗口的透明度用 Alpha 表示，在 DJYGUI 中，其值范围定义在 0 - 255 之间。Alpha 的取值表示窗口的不透明属性，窗口 alpha 取值为 0，表示窗口完全透明，alpha 取值为 255，表示窗口完全不透明，alpha 值越大，越不透明。GK 支持双 Alpha 混合运算，如表 2-1 所示，S 和 D 分别表示混合的两种颜色，As 和 Ad 表示混合的两个 alpha 值，dst 是最终计算的 alpha 值。

表 2-1 alpha 混合运算

混合模式	计算公式	说明
CN_ALPHA_MODE_AsN	dst = S*As+D*(1-As)	
CN_ALPHA_MODE_AsAdN	dst= S*As+D*(1-As)*Ad	
CN_ALPHA_MODE_AsAd	dst = S*As+D*Ad	
CN_ALPHA_MODE_As	dst = S*As	
CN_ALPHA_MODE_NEED_DST	无	bit7=1,需要 dst 像素参与运算
CN_ALPHA_MODE_NEED_AD	无	bit6=1,需要 dst alpha 参与运算

窗口透明色是指当窗口内需要填充的颜色与透明色相同时，窗口显示背景色。透明色是 GK 对 Windows 定义的 Rop2Code 的一个扩展。绘制光栅位图也可以执行设置透明色的操作，待绘制的位图中是透明色的像素在绘制过程中不显示，只显示背景色。

光栅属性 RopMode 是一个 32 位无符号数，各比特的含义如图 2-2 所示。

作为窗口的属性时，忽略 Rop2 使能位，也不支持需要背景 alpha 参与的 alpha 码（CN_ALPHA_MODE_AsAdN 和 CN_ALPHA_MODE_AsAd）。

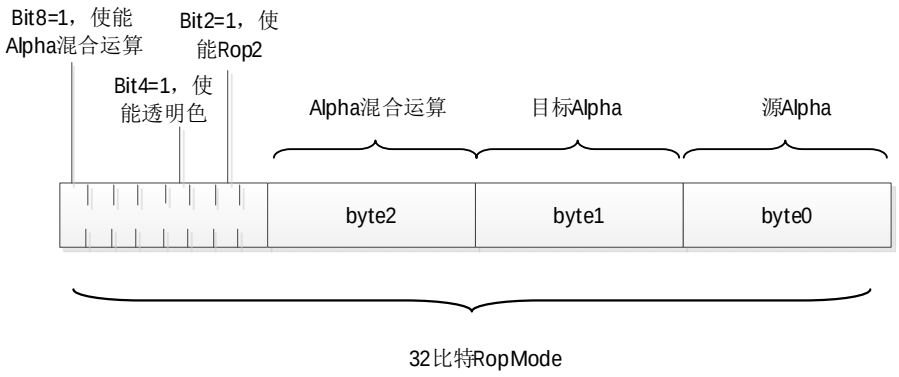

图 2-2 光栅属性设置

2.5. 边界模式

窗口的边界模式是窗口显示边界受限制的属性，它影响窗口的受限边界和显示效果。窗口的边界模式分为受限和不受限两种。如果窗口边界模式为受限，则其边界受限于父窗口的边界限制；如果窗口模式为不受限，窗口将受限于祖先窗口中第一个边界模式为受限窗口的父窗口。

2.6. API 说明

2.6.1. GK_ApiCreateDesktop：创建桌面

bool_t GK_ApiCreateDesktop(struct tagDisplayRsc *Display,
struct gkwin_rsc *Desktop,
s32 Width,s32 Height,u32 Color,
u32 BufMode)

头文件：

gkernel.h

参数：

Display：桌面的宿主显示器。

Desktop：新创建的桌面，保存数据结构所需，内存由调用者提供。

Width：桌面的宽度，小于显示器宽度则调整为显示器宽度。

Height：桌面的高度，小于显示器高度则调整为显示器高度。

Color：创建桌面时填充的颜色，要求颜色格式是真彩色。

BufMode：缓冲模式，见表 3-2。

返回值：

true：创建成功；false：创建失败

说明：

桌面是所有显示器的第一个窗口。每个显示器有且只有一个桌面，display 的成员 desktop 指针指向其桌面。所有的显示器，必须先初始化，成功创建桌面后才能创建窗口，桌面的尺寸不能小于显示器的尺寸，但可以大于显示器尺寸。桌面在刷新 screen 的过程中和普通窗口是等同的。桌面的优先级只能是 0。

2.6.2. GK_ApiGetDesktop：获取桌面

struct tagGkWinRsc *GK_ApiGetDesktop(char *DisplayName)

头文件：

gkernel.h

参数：

DisplayName：桌面所在显示器的名字。

返回值：

桌面指针，为 NULL 时表示未找到该名称桌面。

说明：

在 djygui 中，桌面也是普通窗口，它是第一个建立的窗口，只有建立 desktop 窗口后，才能创建其他窗口。同一个显示器上的所有其他窗口都是桌面的子孙窗口。

2.6.3. GK_ApiCreateGkwin：创建窗口

```
bool_t Gk_ApiCreateGkwin(struct tagGkWinRsc *Parent,
                         struct tagGkWinRsc *NewWin,
                         s32 Left,s32 Top,s32 Right,s32 Bottom,
                         u32 Color,u32 BufMode,
                         char *Name,u16 PixelFormat,u32 KeyColor
                         u32 BaseColor, u32 RopMode)
```

头文件：

gkernel.h

参数：

Parent：父窗口指针。

NewWin：新创建的窗口，保存数据结构所需内存，由调用者提供。

Left：新创建的窗口左边界，相对于父窗口。

Top：新创建的窗口上边界，相对于父窗口。

Right：新创建的窗口右边界，相对于父窗口。

Bottom：新创建的窗口下边界，相对于父窗口。

Color：创建窗口时填充的颜色，要求颜色格式是真彩色。

BufMode：缓冲模式，见表 3-2。

Name：新建窗口名称。

PixelFormat：像素格式，CN_SYS_PF_DISPLAY 表示与显示器相同。

KeyColor：设定窗口透明色，要求颜色格式为真彩色。

BaseColor：灰度基色，(仅在 PixelFormat == CN_SYS_PF_GRAY1 ~8 时有用)。

RopMode：光栅操作码，即二元光栅操作、alpha 和透明色属性。

返回值：

true：窗口创建成功；false：窗口创建失败。

说明：

新创建的窗口只是一个矩形区域，没有填充，也没有边框等一切装饰。新建窗口创建时默认优先级为 0，其优先级可以调整。窗口的尺寸不受限制。新建窗口遗传父窗口的一些属性，如宿主显示器、直接写屏。窗口的像素格式可调，设置为 CN_SYS_PF_DISPLAY 表示与显示器相同。窗口的 RopMode 属性可设置为 0，表示无光栅操作属性，同时可根据 alpha、透明色设置相应属性，但窗口不支持二元光栅。

示例：

```
static struct tagGkWinRsc *desktop,gkwin1;
desktop = GK_ApiGetDesktop("ili9325");
GK_ApiCreateGkwin(desktop,&gkwin1,0,0,200,200,CN_COLOR_RED,
                                    CN_WINBUF_NONE,"window1",
                                    CN_SYS_PF_DISPLAY, CN_COLOR_BLACK,
                                    CN_COLOR_BLACK , 0 );
```

代码中，ili9325 是显示器的名字。

上面的一段程序，在桌面上创建了一个名为 window1 的窗口，填充颜色为红色 CN_COLOR_RED(0xff0000)，缓冲模式是没有缓冲区（CN_WINBUF_NONE），只能进行直接写屏操作，窗口左上角、右下角坐标分别是（0,0）、（200,200），给定坐标是相对于父窗口的，窗口像素格式与显示器一致，窗口透明色和灰度基色为黑色（CN_COLOR_BLACK），窗口 RopMode 属性为 0（无特殊效果）。

2.6.4. GK_ApiDestroyWin：销毁窗口

void GK_ApiDestroyWin(struct tagGkWinRsc *Gkwin)

头文件：

gkernel.h

参数：

Gkwin：目标窗口。

返回值：

无。

说明：

销毁窗口，同时销毁窗口的子孙窗口。

2.6.5. GK_ApiMoveWin：移动窗口

void gk_api_move_win(struct tagGkWinRsc *Gkwin,s32 Left,s32 Top,u32 SyncTime)

头文件：

gkernel.h

参数：

Gkwin：显示的目标窗口。

Left：窗口移动后的左上角坐标的横坐标，相对于父窗口。

Top：窗口移动后的左上角坐标的纵坐标，相对于父窗口。

SyncTime：见 3.2 章节。

返回值：

无。

说明：

改变窗口在父窗口内的相对位置，由于子窗口的坐标是相对于父窗口的，故移动窗口时, 连子窗口一起移动。

2.6.6. GK_ApiSetBoundMode：设置边界模式

void GK_ApiSetBoundMode (struct tagGkWinRsc *Gkwin, bool_t Mode)

头文件：

gkernel.h

参数：

Gkwin：目标窗口。

Mode：窗口的边界模式，true 为受限，false 为不受限。

返回值：

无。

说明：

设定窗口的显示边界是否受父窗口限制，限制后，子窗口超出父窗口的部分将不予显示, 桌面的直接子窗口默认受限，不能更改。

2.6.7. GK_ApiSetPrio：设置窗口显示优先级

void GK_ApiSetPrio(struct tagGkWinRsc *Gkwin, sfast_t Prio,u32 SyncTime)

头文件：

gkernel.h

参数：

Gkwin：目标窗口。

Prio：新优先级。

SyncTime：见 3.2 章节。

返回值：

无。

说明：

设置一个窗口的优先级，优先级决定当前窗口在同级窗口中 z 轴相对位置，数字越小，在 z 轴中越靠前，优先级<=0 将覆盖父窗口，反之被父窗口覆盖。窗口改变优先级后，它在 Z 轴中的顺序可能会改变，屏幕内容也可能会改变。由于被改变优先级的窗口可能还有子窗口，所以，在 z 轴中被移动的，不是一个窗口，而是连续的一组窗口。桌面的优先级为 0，不可改变。窗口设置的新优先级可以和原优先级相同，即便如此，显示效果也可能会改变。新设置的优先级需在合法范围里（-128 - 127）。

2.6.8. GK_ApiSetRopMode：设置光栅属性

bool_t GK_ApiSetRopMode(struct tagGkWinRsc *Gkwin,
u32 RopCode,u32 SyncTime)

头文件：

gkernel.h

参数：

Gkwin：目标窗口。

RopCode：光栅操作码，可以是扩展的光栅操作码，参见图 2-2。

SyncTime：见 3.2 章节。

返回值：

true：设置成功；false：设置失败。

说明：

非缓冲窗口和桌面不可修改光栅操作码。

2.6.9. GK__ApiSetTransparentColor：设置窗口透明色

bool_t GK_ApiSetTransparentColor(struct tagGkWinRsc *Gkwin, u32 KeyColor)

头文件：

gkernel.h

参数：

Gkwin：目标窗口。

KeyColor：要设置的透明色。

返回值：

true：设置成功；false：设置失败。

说明：

桌面不需要设置透明色。

3. 窗口绘制

3.1. 概述

GK 支持的窗口绘制包括单个像素、直线（含/不含端点）、圆、曲线、文本和矩形填充。

同步异步操作是影响了绘图的调用返回方式，而缓冲模式则决定了绘制的过程，像素颜色、二元光栅操作和渐变填充直接影响绘图效果，本章将作详细介绍。

3.2. 同步与异步操作

窗口绘制同步操作是指应用程序调用绘制函数，绘制完成或超时才返回，异步操作是指调用函数后立刻返回，此时绘制操作可能未完成。

绝大多数绘制函数中，都有一个参数 SyncTime，如果该参数为 0，上层应用程序调用绘制函数后，绘制消息发送到消息管道后直接返回，此时，绘制操作可能未完成。若连续多次调用绘制函数，可能导致消息队列拥塞。如果 SyncTime 是一个非零的值，调用线程将被阻塞，直到执行完绘制命令，或者超时才解除阻塞。超时时间由 SyncTime(us)设置。

3.3. 二元光栅操作

二元光栅操作是指通过二元光栅操作码 ROP2，组合画笔或画刷像素和目标像素以得到新的目标像素。

二元光栅操作的本质是位逻辑运算。位逻辑运算将独立的位以参数 D、P 的形式进行配对，同时对位进行布尔逻辑运算。进行二元光栅操作用到的布尔逻辑操作有 AND（&）、OR（|）、NOT（~）、XOR（^）。二元的布尔逻辑运算，有 16 中可能的组合，如表 3-1 所示。

表 3-1 二元光栅操作

二元光栅操作	公式	说明
CN_R2_BLACK	dest = 0	RGB 模式下是黑色
CN_R2_NOTMERGEPEN	dest = ~(dest | pen)	CN_R2_MERGEPEN 取反
CN_R2_MASKNOTPEN	dest = ~pen & dest	画笔颜色取反与目标颜色的逻辑乘
CN_R2_NOTCOPYEN	dest = ~pen	画笔颜色取反
CN_R2_MASKPENNOT	dest = ~dest & pen	目标颜色取反与画笔颜色的逻辑乘
CN_R2_NOT	dest = ~dest	目标颜色取反
CN_R2_XORPEN	dest = dest ^ pen	目标颜色与画笔颜色的逻辑异或
CN_R2_NOTMASKPEN	dest = ~(dest & pen)	CN_R2_MASKPEN 取反
CN_R2_MASKPEN	dest = dest & pen	目标颜色与画笔颜色的逻辑乘
CN_R2_NOTXORPEN	dest = ~(dest ^ pen)	CN_R2_XORPEN 取反
CN_R2_NOP	dest = dest	颜色不变
CN_R2_MERGENOTPEN	dest = ~pen | dest	画笔颜色取反与目标颜色的逻辑和
CN_R2_COPYPEN	dest = pen	画笔的颜色
CN_R2_MERGEPENNOT	dest = ~dest | pen	目标颜色取反与画笔颜色的逻辑和
CN_R2_MERGEPEN	dest = dest | pen	目标颜色与画笔颜色的逻辑和
CN_R2_WHITE	dest = 1	RGB 模式下是白色，不一定是一位

注：表 3-1 中dest（也写作D）、pen（也写作P）分别为二元光栅操作的目标颜色和画笔颜色。

在 GK 中，二元光栅操作码用 Rop2Code 表示，缩写为 R2。一般默认的二元光栅操作码是 CN_R2_COPYPEN，它把目标像素绘制为画笔的颜色。绘制像素、直线、曲线的情况都使用画笔，用到了二元光栅操作。

二元光栅操作在计算机图形学中起很重要的作用。使用 CN_R2_BLACK 把目标像素绘制为黑色，使用 CN_R2_WHITE 把目标像素绘制为白色，使用 CN_R2_NOP 则目标像素颜色不变。由于二元光栅操作是对位进行操作，可以实现有选择的屏蔽目标颜色中的某些位。

3.4. 像素颜色

3.4.1. 像素颜色格式

颜色是人产生的一种对光的视觉效应，通常是由几个颜色属性进行描述。RGB 就是一种对颜色进行描述的模式。

RGB 代表红色（red）、绿色（green）、蓝色（blue）三种颜色，即通常所说的三原色。RGB 图像只使用红、绿、蓝三种颜色，将它们以不同的比例混合，可以得到各式各样的颜色。红、绿、蓝三种颜色，从暗到亮划分为等阶亮度，以 255 阶为例，在 0 时颜色最弱，而在 255 时颜色最亮；当三种颜色都为 255 时，混合显示为白色，当三种颜色都为 0 时，混合显示为黑色，当三种颜色阶数相等且不等于 0、255 时，显示灰色，颜色介于白色与黑色之间；三种颜色阶数不等时，会得到彩色。特殊情况下，单色，即 1 位有两种颜色，0 代表黑色，1 代表白色。灰色，每一位均为 1，则为纯白色，全为 0，则为纯黑色，中间为灰色，且由 0-1 渐变，越来越白。

GK 支持多种像素颜色格式，例如，当前比较流行的 16 位色 RGB565 和 24 位色 RGB888 等。Djygui 支持的颜色格式在 gkernel.h 中定义，参见 CN_SYS_PF_RGB565 族常数。

3.4.2. 像素颜色格式转换

GK 提供的所有接口统一要求绘图颜色格式为真彩色，因此，针对不同的显示器，GK 提供像素颜色格式转化接口，通过颜色格式转换接口，可将真彩色转化为本地颜色格式，或将本地颜色格式转化为真彩色。

3.5. 缓冲模式

GK 窗口支持三种缓冲模式，分别为继承缓冲、有窗口缓冲和无缓冲（即直接写屏）。

在高性能富内存的系统中，比如智能手机有数百 M 甚至上 G 的内存，GK 使用窗口缓冲来提高 GUI 性能，降低应用程序复杂度。在内存资源非常紧张的嵌入式系统中，GK 支持支持直接写屏的方式，降低内存消耗。

帧缓冲是显示器的属性，显示的数据可存储在帧缓冲区，缓冲区大小由显示器大小和像素格式决定。窗口缓冲是窗口显示的缓冲区，它的大小由窗口大小和像素格式决定。只有包含帧缓冲区的情况下，才可以创建带窗口缓冲区的窗口。有窗口缓冲区时一定存在帧缓冲区，有帧缓冲区时不一定存在窗口缓冲区。

GK 创建窗口时可指定窗口缓冲模式，用于指定创建的窗口是否带缓冲区。缓冲模式只能在创建窗口时设定，不能修改。窗口缓冲模式如表 3-2 所示。

表 3-2 窗口缓冲模式

缓冲模式	宏定义	意义
CN_WINBUF_PARENT	0	继承父窗口，若为桌面，则继承帧缓冲
CN_WINBUF_NONE	1	无有缓冲区
CN_WINBUF_BUF	2	有缓冲区

根据窗口的缓冲模式，窗口的绘制过程如图 3-1 所示。

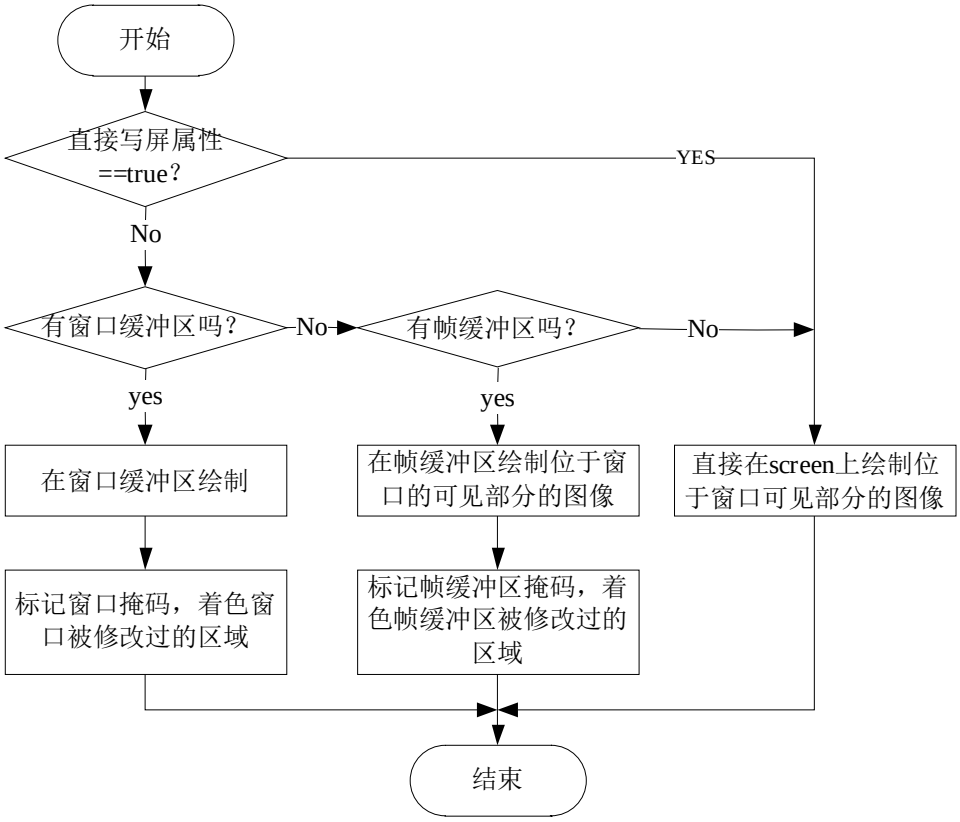

图 3-1 窗口绘制过程

3.6. 渐变填充

GK 渐变填充功能是在指定的矩形区域内，按照一定的填充方式，填充效果由 Color0 到 Color1 渐变的一种特殊的填充方式。按照填充的方式，渐变填充模式如表 3-3 所示。其中 Color0 是渐变填充颜色 0，Color1 是渐变填充颜色 1。

渐变填充过程中，首先求出 Color0 和 Color1 的 alpha 和 R、G、B 分量，然后将 Color0 和 Color1 的分量线性整合，最后将 alpha 值和 R、G、B 合成 Color 的 ARGB 值。

表 3-3 渐变填充模式

模式	数值	说明
CN_FILLRECT_MODE_N	0	直接填充，只有 Color0 有效
CN_FILLRECT_MODE_H	1	水平填充，Color0 表示左边颜色，Color1 右边
CN_FILLRECT_MODE_V	2	垂直填充，Color0 表示上边颜色，Color1 下边
CN_FILLRECT_MODE_SP	3	倾斜填充，Color0 表示左上角颜色，Color1 右下角
CN_FILLRECT_MODE_SN	4	倾斜填充，Color0 表示右上角颜色，Color1 左下角

3.7. API 说明

3.7.1. GK_ApiSetPixel：绘制像素

```
void GK_ApiSetPixel(struct tagGkWinRsc *Gkwin,s32 x,s32 y,
                    u32 Color,u32 Rop2Code,u32 SyncTime)
```

头文件：

gkernel.h

参数：

Gkwin：绘制的目标窗口。

x：绘制位置的横坐标，相对于窗口。

y：绘制位置的纵坐标，相对于窗口。

Color：画点使用的颜色，要求颜色格式是真彩色。

Rop2Code：二元光栅操作码。

SyncTime：见 3.2 章节。

返回值：

无。

3.7.2. GK_ApiLineto：绘制直线

void GK_ApiLineto(struct tagGkWinRsc *Gkwin, s32 x1,s32 y1,
s32 x2,s32 y2,u32 Color,u32 Rop2Code,u32 SyncTime)

头文件：

gkernel.h

参数：

Gkwin：显示的目标窗口。

x1：待绘制直线起点的横坐标，相对于窗口。

y1：待绘制直线起点的纵坐标，相对于窗口。

x2：待绘制直线终点的横坐标，相对于窗口。

y2：待绘制直线终点的纵坐标，相对于窗口。

Color：画线使用的颜色，要求颜色格式是真彩色。

Rop2Code：二元光栅操作码。

SyncTime：见 3.2 章节。

返回值：

无。

说明：

绘制单像素先宽的直线，结束端点不绘制。此外，函数 GK_ApiLinetoIe 也是绘制单像素宽直线，区别在于后者绘制直线的终点。

示例：

画点是画直线的基础，直线是由多个点构成的。下面以画点、画直线的实例介绍画点和画直线的函数的使用方法。

```
static struct tagGkWinRsc *desktop;
desktop = GK_ApiGetDesktop("ili9325");
GK_ApiSetPixel (desktop,0,0,CN_COLOR_RED,CN_R2_COPYEN,,CN_GK_SYNC);
GK_ApiLineto (desktop,0,2,5,2,CN_COLOR_BLUE,CN_R2_CYPYEN, CN_GK_SYNC);
GK_ApiLinetoIe (desktop,0,3,5,3,CN_COLOR_BLACK, CN_R2_CYPYEN,CN_GK_SYNC);
```

在桌面上进行画点、画线操作。上面的程序实现的是在桌面的前四行，第一行是画一个像素，第二行画一条不包含终点的直线，第三行是画一条包含终点的直线。

3.7.3. GK_ApiDrawCircle：绘制圆形

void GK_ApiDrawCircle(struct tagGkWinRsc *Gkwin,s32 x0,s32 y0,
u32 r,u32 Color,u32 Rop2Code,u32 SyncTime)

头文件：

gkernel.h

参数：

Gkwin：绘制的目标窗口。

x0：圆心点的横坐标，相对于窗口。

y0：圆心点的纵坐标，相对于窗口。

r：待绘制圆的半径。

Color：画圆使用的颜色，要求颜色格式是真彩色。

Rop2Code：二元光栅操作码。

SyncTime：见 3.2 章节。

返回值：

无。

说明：

画一个单像素线宽的圆。

3.7.4. GK_ApiDrawText：绘制文本

```
void GK_ApiDrawText(struct tagGkWinRsc *Gkwin,s32 x,s32 y,
                    const char *Text,u32 Count,u32 Color,
                    u32 Rop2Code,u32 SyncTime)
```

头文件：

gkernel.h

参数：

Gkwin：显示的目标窗口。

x：显示位置的横坐标，相对于窗口。

y：显示位置的纵坐标，相对于窗口。

Text：待显示的字符串。

Count：字符串长度，含串结束符。

Color：显示颜色，要求颜色格式是真彩色。

Rop2Code：二元光栅操作码。

SyncTime：见 3.2 章节。

返回值：

无。

说明：

文本串是由默认字体和默认字符集显示的，调用此函数显示的是单色的字，一个汉字为两个字符。

3.7.5. GK_ApiBezier：绘制贝塞尔曲线

```
void gk_api_Bezier(struct tagGkWinRsc *Gkwin,float x1,float y1,
                   float x2,float y2,float x3,float y3,float x4,float y4,
                   u32 Color,u32 Rop2Code,u32 SyncTime)
```

头文件：

gkernel.h

参数：

Gkwin：显示的目标窗口。

x1：贝塞尔曲线第一个控制点的横坐标，相对于窗口。

y1：贝塞尔曲线第一个控制点的纵坐标，相对于窗口。
x2：贝塞尔曲线第二个控制点的横坐标，相对于窗口。
y2：贝塞尔曲线第二个控制点的纵坐标，相对于窗口。
x3：贝塞尔曲线第三个控制点的横坐标，相对于窗口。
y3：贝塞尔曲线第三个控制点的纵坐标，相对于窗口。
x4：贝塞尔曲线第四个控制点的横坐标，相对于窗口。
y4：贝塞尔曲线第四个控制点的纵坐标，相对于窗口。
Color：画贝塞尔曲线使用的颜色，要求颜色格式是真彩色。
Rop2Code：二元光栅操作码。
SyncTime：见 3.2 章节。

返回值：

无。

说明：

画单像素宽的三次 Bezier 曲线。贝塞尔曲线是电脑图形学中相当重要的参数曲线，可以用来绘制各种形状的曲线，djygui 提供绘制 3 次贝塞尔曲线的 api 函数。

示例：

以特殊情况，圆和贝塞尔曲线都在窗口内的实例来说明画圆与画贝塞尔曲线函数的使用。

```
static struct tagGkWinRsc *desktop;
desktop = GK_ApiGetDesktop("ili9325");
GK_ApiDrawCircle(desktop,100,100,50,CN_COLOR_RED, CN_R2_COPYPEN,CN_GK_SYNC);
GK_ApiBezier(desktop,0,200,50,50,150,20,300,200,
                CN_COLOR_RED,CN_R2_COPYPEN,CN_GK_SYNC);
```

实例显示的效果是在桌面上有一个圆心在（100,100），半径为 50，红色的圆，同时有一条起点为（0,200），终点为（300,200）的红色抛物线。

3.7.6. GK_ApiFillWin：填充窗口

void GK_ApiFillWin(struct tagGkWinRsc *Gkwin,u32 Color,u32 SyncTime)

头文件：

gkernel.h

参数：

Gkwin：待填充的窗口。
Color：给定的颜色，要求颜色格式是真彩色。
SyncTime：见 3.2 章节。

返回值：

无。

说明：

填充窗口是用给定颜色填充整个窗口，颜色格式要求为真彩色。

3.7.7. GK_ApiFillRect：填充矩形

void GK_ApiFillRect(struct tagGkWinRsc *Gkwin, struct tagRectangle *Rect,
u32 Color0, u32 Color1, u32 Mode, u32 SyncTime)

头文件：

gkernel.h

参数：

Gkwin：待填充的窗口。
Rect：待填充的位置，坐标相对于待填充的窗口。

Color0：颜色 0，要求颜色格式是真彩色。

Color1：颜色 1，要求颜色格式是真彩色。

Mode：渐变模式，见表 3-3。

SyncTime：见 3.2 章节。

返回值：

无。

说明：

填充窗口一部分，一部分是指窗口中的任一矩形域，只会填充到可视域内。填充模式是指可填充渐变效果。

示例：

GK_ApiFillRect 填充指定矩形范围，支持渐变填充。下面举例说明这个函数的使用方法。

```
static struct tagGkWinRsc gkwin1;
GK_ApiFillRect (&gkwin1,rect,CN_COLOR_GREEN, CN_COLOR_BLUE,
                                              ,CN_FILLRECT_MODE_H,CN_GK_SYNC);
```

这段程序对桌面上窗口进行矩形渐变填充，填充颜色 0 为绿色，填充颜色 1 为蓝色，采用的渐变模式为水平填充，填充效果为窗口的矩形区域 rect 内，从左到右的内容由绿色变为蓝色。

3.7.8. GK_ApiSyncShow：刷新窗口

void GK_ApiSyncShow(u32 SyncTime)

头文件：

gkernel.h

参数：

SyncTime：见 3.2 章节。

返回值：

无。

说明：

执行该命令，所有输出数据将输出到显示器。

3.7.9. GK_ConvertColorToRGB24：将本地颜色转化为真彩色

u32 GK_ConvertColorToRGB24(u16 PixelFormat,u32 Color,ptu32_t ExColor)

头文件：

gk_draw.h

参数：

PixelFormat：本地颜色格式。

Color：源色彩。

ExColor：扩展颜色。

返回值：

真彩色格式的颜色。

说明：

颜色格式转换，把一个本地格式颜色转换成ERGB8888，如果位图格式如果包含alpha通道，则先计算alpha。ExColor为双功能参数，如果PixelFormat=CN_SYS_PF_GRAY1~8，或者PixelFormat=CN_SYS_PF_ALPHA1~8，则ExColor表示基色。如果PixelFormat=CN_SYS_PF_PALETTE1，则ExColor表示调色板指针。如果PixelFormat不需要此参数，可给0或NULL。

3.7.10. GK_ApiSetDirectScreen：设置窗口直接写屏

void GK_ApiSetDirectScreen(struct tagGkWinRsc *gkwin, u32 SyncTime)

头文件：

gkernel.h

参数：

gkwin：目标窗口。

SyncTime：见 3.2 章节。

返回值：

无。

说明：

设置窗口直接写屏属性，当窗口直接写屏属性被设置为 true 时，无论窗口是否有缓冲，所有绘制操作都是直接绘制到显示器上面。

3.7.11. GK_ApiCancelDirectScreen：撤销窗口直接写屏

void GK_ApiCancelDirectScreen(struct tagGkWinRsc *Gkwin, u32 SyncTime)

头文件：

gkernel.h

参数：

Gkwin：目标窗口。

SyncTime：见 3.2 章节。

返回值：

无。

说明：

撤销窗口直接写屏属性。

3.7.12. GK_ConvertRGB24ToPF：将真彩色转化为本地颜色

u32 GK_ConvertRGB24ToPF(u16 PixelFormat,u32 Color)

头文件：

gk_draw.h

参数：

PixelFormat：目标颜色格式。

Color：源色彩，颜色格式为真彩色。

返回值：

目标颜色格式的色彩。

说明：

把一个 ERGB8888 格式的颜色转换成本地格式，本地格式不允许有 alpha 通道，也不允许是调色板位图。

4. 文本显示

4.1. 概述

计算机使用二进制语言，而使用计算机的人则用各自的自然语言，这些自然语言的文字，需要编码才能在计算机中标识。为了让计算机“理解”人类的语言，人们设计了许多字符集和字符编码方式，主要分以下两大类。

1. ansi 系

ansi 系是一种兼容 ascii 编码的多字节字符集，其显著特点是：

兼容编码值为 0x00~0x7f 的单字节 ascii 编码，任何 ascii 编码的文本，都是合法的字符串。正常字符流中不会出现值为 0x00 的字节，这点很重要，因为 c 语言把 0x00 看成字符串结束符。

字符是变长的，一个字符可能是 1 字节、2 字节、3 字节甚至 4 字节，多字节字符的首字节最高位必为 0。

一个字符集只能表示一种语言，不同的 ansi 系的字符集，可能有冲突，即不同语言的不同字符，编码可能相同。

2. unicode 系

unicode 是一种收录了以及打算收录世界上所有字符的单一编码字符集，因此，它跟 ansi 系不一样，unicode 只有一种字符集。unicode 把字符分为 17 个页面，每个页面 65536 个编码，页面 0 也叫 BMP 页面，我们常用的绝大多数字符都在 BMP 页面上。unicode 的字符集和字符编码是分开的，主要字符编码有：

ucs-2：只能实现 unicode 的一个子集的编码，只能表示 BMP 页面上的字符，所有字符都编码成 2 字节，在正常字符流中有 0x00 这个字符，需特别注意。

ucs-4：能实现所有 unicode 字符，所有字符都编码成 2 字节，同样在正常字符流中有 0x00 这个字符，需特别注意。

utf-8：能实现所有 unicode 字符，是一种变长字符编码，最长者可达 6 字节，但按照目前 unicode 的实际标准，表示 unicode 定义的所有 17 个页面的字符，只需要 4 个字节，5、6 字节编码其实要等猴年马月才能用上。utf-8 的好处是，正常字符流中间不会出现 0x00

utf-16：同样是一种变长编码，用 2 字节或 4 字节来表示一个字符，绝大多数 BMP 页面上的字符可以用 2 字节编码来表示，超出 2 字节的部分用 4 字节来表示。同样，其正常字符流中有 0x00 这个字符，需特别注意。

DJYOS 的语言支持是多方面的，这里只描述与图形界面输出相关的部分。

djygui 提供多语言支持，允许用户设计多语言支持界面。djygui 中，字符集和字体都作为一种资源存在，应用程序通过选择相应的资源来支持多国语言。应用程序也可以创建资源来实现自己的“小语言”，例如，某应用程序只需要使用 100 个汉字，且存储空间紧张，就可以创建一个小语言，该语言只包含 100 个汉字。

语言属性由字符集和字体资源组成，字符集资源决定本字符集能表示多少种字符和字符如何编码；字体资源决定本字体有多少种字符和每个字符的字形。目前 DJYOS 提供了 ascii6x12 字体、ascii8x16 字体、ascii8x8 字体、gb2312 16 点阵字体等字体可供选择使用。同时 DJYOS 支持 ASCII、Unicode、GB2312 等字符编码。

4.2. 字符集说明

字符集资源是用来映射一个字符集合中的字符与二进制代码之间的转换关系的，一个字符集可以表示一种语言、多种语言，或者是一种语言的一个子集，甚至是所有 unicode 字符，视字符集的定义而定。在 DJYOS 中，字符集资源并非只为 gui 服务，这里只介绍跟 gui 有关的部分。

正如上文提及 DJYOS 支持 ASCII、utf8、GB2312 等字符编码，但是 DJYOS 将 ucs4 作为桥梁，ASCII、utf8、GB2312 等字符编码首先转换成 ucs4，进而进行编码显示。要想将一个字符编码转换成 ucs4，需要完成以下工作：

字符解析函数，把给定的字符转化成 ucs-4 编码；

字符串解析函数，从一个字符串中取出一个字符并转化为 ucs-4 编码，并给出这个字符的长度（字节数）；

逆字符解析函数，把一个 ucs-4 编码的字符转换成本字符集的字符，并给出这个字符的长度（字节数）。

字符编码结构体 tagCharEncodingRsc 用于表征某个字符编码，其定义如下：

```
struct tagCharEncodingRsc
{
    struct tagRscNode node;
    /* 单个字符的最大字节数 */
    u32 max_len;
//----多字节字符转为ucs4字符
//功能: 把一个多字节字符转换为ucs4字符。
//参数: pwc，保存转换结果的指针，调用方确保不为NULL,不判断，gui将确保本函数不会被其他模//块调用;
//      mbs, 指向待转换的多字节字符字节序列的指针(由调用函数判断s的合法性);
//      n，最大检测长度。
//返回: 0: pwc,mbs是NULL指针或者mbs指向空串。
//      -1:mbs指向的不是合法多字节字符，或者长度n内未能检测到完整多字节字符
//      其他:mbs缓冲区内第一个完整多字节字符的长度。
//说明:
//      此函数是C Run-time库的mblen及mbtowc服务函数。
//      传入的s指针必须非NULL
    s32 (*mb_to_ucs4)(u32* pwc, const char* mbs, s32 n);
//----ucs4字符转为多字节字符
//功能: 把一个ucs4字符转换为多字节字符。
//参数: mb，保存转换结果的指针(由调用函数判断s的合法性)
//      wc，待转换的字符
//返回: -1,如果wc不能对应一个有效的多字节字符，
//      字节数，返回对应的多字节字符的字节数。
//说明:
//      此函数是C Run-time库的wctomb服务函数。
    s32 (*ucs4_to_mb)(char* mb, s32 wc);
//----多字节字符串转为ucs4串
//功能: 把一个多字节字符串转换为ucs4字符串
//参数: pwcs，保存转换结果的指针，缓冲区由调用方提供，若空，则本函数转变为只计算
//          保存转换结果所需的字节数
//      mbs，保存多字节字符的缓冲区指针
//      n，最大检测长度，
//返回: 0: mbs是NULL指针
//      -1:结束条件到达前，有不能转换的字符
//      其他:得到的字符数，=n表示源串是不包含串结束符'\0'。
    s32 (*mbs_to_ucs4s)(u32* pwcs, const char* mbs, s32 n);
//----ucs4字符串转为多字节字符串----------------------------------------------
//功能: 把一个ucs4字符串转换为多字节字符串。
//参数: mbs，保存转换结果的指针，缓冲区由调用方提供，若空，则本函数转变为只计算保存转换结//果所需的
字节数
//      pwcs，待转换的字符
//      n，最大检测长度，遇串结束符或长度达到n结束转换，注意ucs4的结束符是连续4个0x00.
//返回: 0: pwcs是NULL指针
//      -1:结束条件到达前，有不能转换的字符
//      其他:写入mbs缓冲区的字节数，含串结束符'\0'
//-----------------------------------------------------------------------------
    s32 (*ucs4s_to_mbs)(char* mbs, const u32* pwcs, s32 n);
};
```

要想在 DJYOS 上使用某个特定字符编码，其实就是实现该字符编码结构体中 4 个钩子函数，使其能与 ucs4 进行转换。如果实现了相应的转换函数，将其挂接到其字符编码结构体中，然后将其安装到系统字符编码资源树下，即可实现支持该字符编码。例如 DJYOS 支持 gb2312，需要首先调用系统函数 Charset_CharEncodingModuleInit 创建字符编码树资源节点（如果已创建则不必调用），然后调用系统函数 Charset_Gb2312EncodeModuleInit 将该字符编码安装到字符编码树节点下即可。Charset_Gb2312EncodeModuleInit 函数源代码如下：

```
ptu32_t Charset_Gb2312EncodeModuleInit(ptu32_t para)
{
    static struct tagCharEncodingRsc encoding;
```

```
    encoding.max_len = 2;
    encoding.mb_to_ucs4 = __Gk_Gb2312MbToUcs4;  //多字节字符转为ucs4字符
    encoding.ucs4_to_mb = __Gk_Gb2312Ucs4ToMb;  // ucs4字符转为多字节字符
    encoding.mbs_to_ucs4s = __Gk_Gb2312MbsToUcs4s;// 多字节字符串转为ucs4串
    encoding.ucs4s_to_mbs = __Gk_Gb2312Ucs4sToMbs;// ucs4字符串转为多字节字符串
    if(Charset_NlsInstallCharEncoding(&encoding, CN_NLS_CHARSET_GB2312))
    {
        printf("gb2312 encoding install sucess\n\r");
        return 1;
    }else
    {
        Djy_SaveLastError(EN_GK_CHARSET_INSTALL_ERROR);
        printf("gb2312 encoding install fail\n\r");
        return 0;
    }
}
```

除了标准字符集外，应用程序也可以方便地安装自己定义的字符集，例如某项目只用到 100 个汉字和 50 个日文字符，如果要节省存储空间，完全可以自己创建一个名字为 cnjp 的字符集，包含 100 个汉字和 50 个日文字符。只需将该字符编码安装到 DJYOS 字符编码根资源树下即可，安装过程同上述 Gb23123，在此不赘述。

4.3. 字符集 API 说明

4.3.1. Charset_ModuleInit：字符编码模块初始化

ptu32_t Charset_ModuleInit(ptu32_t para)

头文件：

charset.h

参数：

无。

返回值：

成功初始化则返回 1；失败则返回 0。

说明：

字符编码模块初始化，其实就是在资源链中添加字符编码资源根节点。

4.3.2. Charset_NlsInstallCharset：安装字符编码

bool_t Charset_NlsInstallCharset(struct tagCharset *encoding, char* name)

头文件：

charset.h

参数：

encoding，新增的字符编码指针；

name，新增字符编码名。

返回值：

成功安装字符编码则返回 true；失败则返回 false。

说明：

将新字符编码安装到系统字符编码资源根节点下。

4.3.3. Charset_NlsGetCurCharset：获取当前字符编码

struct tagCharset* Charset_NlsGetCurCharset (void)

头文件：

charset.h

参数：

空。

返回值：

当前使用的字符编码结构体指针。

说明：

获取当前使用的字符编码。

4.3.4. Charset_NlsSetCurCharset：设定当前字符编码

struct tagCharset* Charset_NlsSetCurCharset(struct tagCharset* encoding)

头文件：

charset.h

参数：

encoding，设定的字符编码结构体指针。

返回值：

设定后当前字符编码结构体指针。

说明：

把新字符编码设为当前使用的字符编码，新字符编码必须事先安装到系统中。

4.3.5. Charset_NlsSearchCharset：搜索字符编码资源

struct tagCharset* Charset_NlsSearchCharset(const char* name)

头文件：

charset.h

参数：

name，指定的字符编码。

返回值：

如果成功搜到指定字符编码则返回该字符编码结构体指针；否则返回 NULL。

说明：

从字体资源根节点下搜索指定名称的字符编码资源。

4.3.6. Charset_NlsModuleInit：初始化 Nls 模块

ptu32_t Charset_NlsModuleInit(const char * para)

头文件：

nls.h

参数：

无。

返回值：

成功初始化则返回 1；失败则返回 0。

说明：

Nls 模块初始化，其实就是找到本地字符编码资源并将其设定为当前字符编码。

4.3.7. Charset_NlsGetLocCharset：获取区域字符编码

struct tagCharset* Charset_NlsGetLocCharset(const char* loc)

头文件：

nls.h

参数：

loc，区域名称编码。

返回值：

返回匹配的字符编码结构体指针，若未有匹配编码，使用默认的 ASCII 编码。

说明：

根据区域名称获取字符编码，"C"是默认编码的代号。

4.3.8. Charset_AsciiModuleInit：安装 ASCII 字符集

ptu32_t Charset_AsciiModuleInit(ptu32_t para)

头文件：

ascii.h

参数：

para，无实际意义。

返回值：

1=成功，0=失败。

说明：

安装 ascii 字符集，当系统使用西方字符界面时，使用这个字符集。特别注意，gb2312 已经包含了英文字符集，使用中文或中英文混合界面的，不需要安装 ascii 字符集。但是，由于 GB2312 的字库只包含了全角的英文字符，故还需要安装 ascii 的字体资源，尺寸(8*8、8*16)可选。

4.3.9. Charset_Gb2312ModuleInit：安装 Gb2312 字符集

ptu32_t Charset_Gb2312ModuleInit(ptu32_t para)

头文件：

gb2312.h

参数：

para，无实际意义。

返回值：

1=成功，0=失败。

说明：

安装 gb2312 字符集，当系统使用西方字符界面时，使用这个字符集。特别注意，gb2312 已经包含了英文字符集，使用中文或中英文混合界面的，不需要安装 ascii 字符集。

4.3.10. Charset_Utf8ModuleInit：安装 utf8 字符集

ptu32_t Charset_Utf8ModuleInit(ptu32_t para)

头文件：

utf8.h

参数：

para，无实际意义。

返回值：

1=成功，0=失败。

说明：

安装 utf8 字符编码解析器，执行 ucs4 和编码和 utf8 编码之间转换。

4.4. 字体说明

字体资源其实叫字形资源更加贴切，该资源描述了一个集合的字符的图形表示，与字符集一样，字体资源可以包含一种语言、多种语言，或者是一种语言的一个子集，甚至是所有 unicode 字符。一般来说，一个系统使用的字体资源与字符集资源保持一致即可。在 djyos 中，字体资源并非只为 gui

服务，这里只介绍跟 gui 有关的部分。

字体结构体 tagFontRsc 定义如下：

```
struct tagFontRsc
{
    struct  tagRscNode node;
    s32 MaxWidth;       //最宽字符的宽度,纵向显示时可用作为竖行宽
    s32 MaxHeight;      //最高字符的高度
    u32 Attr;
    bool_t (*LoadFont)(void *zk_addr);      //加载字体
    void (*UnloadFont)(void);      //卸载字体
// 获取ucs4编码为charcode字符的显示点阵，把点阵填充到font_bitmap中，调用者应该提
// 供font_bitmap所需内存。
// 如果font_bitmap参数中的bm_bits参数为NULL，则本函数退化为查询函数。在
// font_bitmap参数中返回图像参数，利用它可以计算bm_bits所需内存尺寸，这在一个文
// 档中使用多种尺寸的文字时，特别有用。
// resv,保留参数。
// 返回: true = 成功执行，false=不支持字符
    bool_t (*GetBitmap)(u32 charcode, u32 size, u32 resv,
                        struct tagRectBitmap *font_bitmap);
    s32 (*GetCharWidth)(u32 Charcode);      //获取某字符宽度
    s32 (*GetCharHeight)(u32 CharCode);     //获取某字符高度
};
```

DJYOS 目前支持提供了 ascii6x12、ascii8x16、ascii8x8、gb2312 16 点阵等字体可供选择使用，要想使用某个具体字体只需调用改字体的初始化函如（如 Font_Ascii8x16FontModuleInit），该初始化函数将该字体安装到系统字体资源树节点下。

如果想使用自己创建的字体，那么过程同创建自己的字符编码过程类似，首先要实现字体数据结构中 load_font、unload_font、get_char_bitmap、GetCharWidth、GetCharHeight5 个钩子函数，其中 load_font、unload_font 两个函数可以置空。然后将该字体安装到系统字体资源树节点下即可。

4.5. 字体 API 说明

4.5.1. Font_ModuleInit：字体模块初始化

ptu32_t Font_ModuleInit(ptu32_t para)

头文件：

font.h

参数：

无。

返回值：

成功初始化则返回 true；失败则返回 false。

说明：

字体模块初始化，其实就是在资源链中添加字体资源根节点。

4.5.2. Font_InstallFont：安装字体

bool_t Font_InstallFont(struct tagFontRsc *font, char* name)

头文件：

font.h

参数：

font，新增的字体资源指针；

name，新增字体名。

返回值：

成功安装字体则返回 true；失败则返回 false。

说明：

将新字体安装到系统字体资源根节点下。

4.5.3. Font_GetCurFont：获取当前字符编码

struct tagFontRsc*　Font_GetCurFont(void)

头文件：

font.h

参数：

空。

返回值：

当前字体结构体指针。

说明：

获取当前使用的字体。

4.5.4. Font_SetCurFont ：设定当前字体

struct tagFontRsc*　Font_SetCurFont(struct tagFontRsc* font)

头文件：

font.h

参数：

font，设定的字体结构体指针。

返回值：

设定后当前字体结构体指针。

说明：

把新字体设为当前使用的字体，新字体必须事先安装到系统中。

4.5.5. Font_SearchFont：搜索字体资源

struct tagFontRsc* Font_SearchFont(const char* name)

头文件：

font.h

参数：

name，指定的字体名称。

返回值：

如果成功搜到指定字体则返回该字体结构体指针；否则返回 NULL。

说明：

从字体资源根节点下搜索指定名称的字体资源。

4.5.6. Font_GetFontLineHeight:取字体点阵行高

s32 Font_GetFontLineHeight(struct tagFontRsc* font)

头文件：

font.h

参数：

font，被查询的字体。

返回值：

该字库竖式排版时的行宽(像素值)。

说明：

获取字体中竖行排版行宽，一般是该字体中最宽的那个字符的允许宽度。

4.5.7. Font_GetFontLineWidth：取字体点阵竖行宽

s32 Font_GetFontLineWidth(struct tagFontRsc* font)

头文件：

font.h

参数：

font，被查询的字体。

返回值：

该字库最高的那个字符的宽度(像素值)。

说明：

获取字体中字符的点阵竖行宽度，即该字体中最宽的那个字符的宽度。

4.5.8. Font_GetFontAttr：取字体属性字

s32 Font_GetFontAttr(struct tagFontRsc* font)

头文件：

font.h

参数：

font，被查询的字体。

返回值：

字体属性字，font.c 模块并不解析该属性字。

说明：

获取字体的属性字。

4.5.9. Font_Ascii6x12FontModuleInit：安装 ascii6x12 字体

ptu32_t　Font_Ascii6x12FontModuleInit(ptu32_t para)

头文件：

ascii6x12.h

参数：

空。

返回值：

1=成功，0=失败。

说明：

安装 ascii 字体，当系统使用西方字符界面时，使用这个字符集。特别注意，gb2312 已经包含了英文字体，在使用中文的界面中可以不安装 ascii 字体。

4.5.10. Font_Ascii8x16FontModuleInit：安装 ascii8x16 字体

ptu32_t Font_Ascii8x16FontModuleInit(ptu32_t para)

头文件：

ascii8x16.h

参数：

空。

返回值：

1=成功，0=失败。

说明：

安装 ascii 字体，当系统使用西方字符界面时，使用这个字符集。特别注意，gb2312 已经包含了英文字体，在使用中文的界面中可以不安装 ascii 字体。

4.5.11. Font_Ascii8x8FontModuleInit：安装 ascii8x8 字体

ptu32_t Font_Ascii8x8FontModuleInit(ptu32_t para)

头文件：

ascii8x8.h

参数：

空。

返回值：

1=成功，0=失败。

说明：

安装 ascii 字体，当系统使用西方字符界面时，使用这个字符集。特别注意，gb2312 已经包含了英文字体，在使用中文的界面中可以不安装 ascii 字体。

4.5.12. Font_Gb2312_816_1616_ArrayModuleInit：安装 ascii8x8 字体

ptu32_t Font_Gb2312_816_1616_ArrayModuleInit(ptu32_t para)

头文件：

gb2312_16.h

参数：

文件名。

返回值：

1=成功，0=失败。

说明：

安装 gb2312 16 点阵，字模保存在文件中，文件名由参数传入。

5. 显示器接口

5.1. 概述

显示器是图形显示的终端，图形的所有操作都会直接或间接的体现在显示器上面。DJYGUI 支持多显示器、虚显示器和镜像显示器的功能。应用程序在调用 API 函数绘图前，需安装显示器，按

照 GK 显示器标接口实现驱动函数。

GK 的底层硬件标准接口函数大体分为三类，第一类是在位图中绘图，第二类是在屏幕上绘图，第三类是显示器的控制函数。安装显示器时，将这三类接口函数注册到图形系统，当用户调用 GUI Kernel API 时，这些驱动函数将以回调函数的方式被调用。

5.2. 镜像显示

镜像显示就是在远端计算机上显示一个与本地显示器完全一样的画面，可用于实现远程协助和远程维护。远程操作多是通过通信口进行的，主要有网络、USB、串口等。

要使用远程显示功能，必须在本地先安装一个显示器，然后把远程显示器当作该显示器的镜像显示器，GK 提供 API 函数用于添加镜像显示器。

值得注意的是，本地显示器可以是实际存在的，也可以是虚拟的。对于实际存在的显示器，需要编写完整的显示 driver，对于虚拟的，则只需要提供一个框架，driver 接口中要求的接口函数，均可以不实现。与本地显示器一样，镜像显示器同样需要在本地实现一个 display 驱动，但不必实现所有接口函数，只需要实现显示器接口函数集中的四个函数：

- set_pixel_screen：在屏幕上绘制一个像素
- fill_rect_screen：填充一块矩形屏幕区域
- bm_to_screen：把 bitmap 输出到屏幕
- line_screnn：在屏幕上面画一根线

所有的绘制操作，最后都会在 GK 中变成对 screen 的输出操作，GK 在执行对 screen 操作的同时，调用上述四个函数之一，把绘制命令传送到远端显示器，远端计算机在自己的本地显示器上实现这几个函数的功能。

镜像显示器驱动还需要图像传输协议的配合，典型的是 VNCServer，该协议在 DJYOS 上有移植。使用 VNCServer 的话，远端显示器只需要安装一个 VNC 客户端就可以了。

5.3. 硬件接口

5.3.1. set_pixel_bm：位图中画像素

bool_t (*set_pixel_bm)(struct tagRectBitmap *Bitmap, s32 x,s32 y,u32 Color,u32 Rop2Code);

参数：

Bitmap：目标位图。

x：绘图位置的横坐标，相对于位图。

y：绘图位置的纵坐标，相对于位图。

Color：画图使用的颜色，颜色格式为真彩色。

Rop2Code：二元光栅操作码。

返回值：

true：成功；false：失败。

说明：

在矩形位图中画一个像素，若显示器使用 CN_CUSTOM_PF（自定义）格式，或者有硬件加速功能，应该实现这个函数，否则直接返回 false。即使有硬件加速，但该加速功能不支持 r2_code 编码的话，也返回 false。color 的格式是 CN_SYS_PF_ERGB8888。

5.3.2. line_bm：位图中画直线

bool_t (*line_bm)(struct tagRectBitmap *Bitmap,struct tagRectangle *Limit,
s32 x1,s32 y1,s32 x2,s32 y2,u32 Color,u32 Rop2Code);

参数：

Bitmap：目标位图。

Limit：限制矩形，只允许在该矩形内绘制。

x1：绘图起点的横坐标，相对于位图。

y1：绘图起点的纵坐标，相对于位图。

x2：绘图终点的横坐标，相对于位图。

y2：绘图终点的纵坐标，相对于位图。

Color：画图使用的颜色，颜色格式为真彩色。

Rop2Code：二元光栅操作码。

返回值：

true：成功；false：失败。

说明：

在矩形位图中画一条任意细直线，不含端点，若显示器使用 CN_CUSTOM_PF（自定义）格式，或者有硬件加速功能，应该实现这个函数，否则直接返回 false。即使有硬件加速，但该加速功能不支持 r2_code 编码的话，也返回 false。color 格式是 CN_SYS_PF_ERGB8888。

另外有公交车中画直线函数 line_bm_ie，与 line_bm 的唯一区别在于，line_bm_ie 画直线包括端点，而 line_bm 不包含端点。

5.3.3. fill_rect_bm：位图中填充矩形

```
bool_t (*fill_rect_bm)(struct tagRectBitmap *DstBitmap,
                       struct tagRectangle *Target,
                       struct tagRectangle *Focus,
                       u32 Color0,u32 Color1,u32 Mode);
```

参数：

DstBitmap：目标位图。

Target：目标矩形，填充范围不能超过该矩形。

Focus：聚焦矩形，当前填充矩形区域。

Color0：渐变颜色 0，颜色格式为真彩色。

Color1：渐变颜色 1，颜色格式为真彩色，模式为直接填充时无效。

Mode：填充模式，请参照 gkernel.h 宏定义组 CN_FILLRECT_MODE_N。

返回值：

true：成功；false：失败。

说明：

在矩形位图中填充矩形，若显示器使用 CN_CUSTOM_PF（自定义）格式，或者有硬件加速功能，应该实现这个函数，否则直接返回 false。color 格式是 CN_SYS_PF_ERGB8888。目标矩形 Target 是渐变填充的整个矩形区域，聚焦矩形 Focus 是当前填充的可视域。Target 和 Focus 的关系如图 5-1 所示。如需要填充 Target 内的三个 focus 区域，需要调用该函数三次，每次使用相应的 Focus 参数。

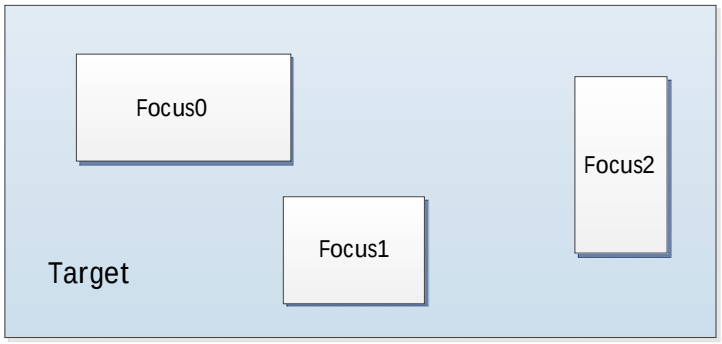

图 5-1 Target 和 Focus 关系图

5.3.4. blt_bm_to_bm：位图中显示矩形

```
bool_t (*blt_bm_to_bm)( struct tagRectBitmap *DstBitmap,
                        struct tagRectangle *DstRect,
                        struct tagRectBitmap *SrcBitmap,
                        struct tagRectangle *SrcRect,
                        u32 RopCode,u32 TransparentColor);
```

参数：

DstBitmap：目标位图。

DstRect：目标矩形。

SrcBitmap：源位图。

SrcRect：源矩形。

RopCode：光栅操作码。

TransparentColor：透明色，为真彩色。

返回值：

true：成功；false：失败。

说明：

在两个矩形位图中位图块传送，如果显示器使用的像素格式是 CN_CUSTOM_PF，或者硬件 2d 加速支持位图块传送，需实现这个函数，否则直接返回 false。对于有硬件加速，但部分 RopCode 编码不支持的情况，可以实现支持的 RopCode，不支持的部分，返回 false。本函数返回 false 的话，gui kernel 会自行用逐像素方法拷贝。目标矩形和源矩形相对坐标不同，但大小相同。

5.3.5. get_pixel_bm：位图中读取像素

```
u32 (*get_pixel_bm)(struct tagRectBitmap *Bitmap,s32 x,s32 y);
```

参数：

Bitmap：目标位图。

x：读像素的横坐标，相对于目标位图。

y：读像素的纵坐标，相对于目标位图。

返回值：

读取的像素值，需转化为真彩色。

说明：

从矩形位图中取一像素，并转换成 CN_SYS_PF_ERGB888。只有在 bitmap 的像素格式为 CN_CUSTOM_PF 时，才需要读取。如果显卡不打算支持自定义格式，本函数直接返回 0。

5.3.6. get_rect_bm：位图中剪切矩形

bool_t (*get_rect_bm)(struct tagRectBitmap
*Src, struct tagRectangle *Rect, struct tagRectBitmap *Dest);

参数：

Src：源位图。

Rec：读取位图的矩形区域。

Dest：目标位图。

返回值：

true：成功；false：失败。

说明：

把 src 位图内 rect 矩形区域的内容复制到 dest 位图中，调用前，先设置好 dest 的 FixelFormat，本函数不理会 src 的 FixelFormat，直接使用 screen 的格式。本函数用于从窗口剪切矩形，而 blt_bm_to_bm 用于显示矩形。如果显示器使用的像素格式是 CN_CUSTOM_PF，或者硬件加速支持位图间拷贝图形，需实现这个函数，否则直接返回 false。由于 dest->FixelFormat 存在很多可能，即使有硬件加速，也存在只支持部分 FixelFormat 的情况，对不支持的格式，返回 false。

5.3.7. set_pixel_screen：屏幕上画像素

bool_t (*set_pixel_screen)(s32 x,s32 y,u32 Color,u32 Rop2Code);

参数：

x：绘制像素的横坐标，相对于屏幕。

y：绘制像素的纵坐标，相对于屏幕。

Color：绘制像素的颜色，为真彩色。

Rop2Code：二元光栅操作码。

返回值：

true：成功；false：失败。

说明：

在 screen 中画一个像素，有 frame buffer 的情况下，正常显示 gui 不会调用这个函数，如果窗口 direct_screen==true，则可能调用本函数。因此，无论是否有 frame buffer，driver 都必须提供并且必须实现本函数。镜像显示器的 driver 必须实现本函数。color 的格式是 CN_SYS_PF_ERGB8888，因此绘制前需转化为本地颜色格式。

5.3.8. line_screen：屏幕上画直线

bool_t (*line_screen)(tagRectangle *limit, s32 x1,s32 y1,s32 x2,s32 y2,
u32 Color,u32 Rop2Code);

参数：

limit：限制矩形，只绘制矩形内部的部分。

x1：绘图起点横坐标，相对于屏幕。

y1：绘图起点纵坐标，相对于屏幕。

x2：绘图终点横坐标，相对于屏幕。

y2：绘图终点横坐标，相对于屏幕。

Color：绘制像素的颜色，为真彩色，因此绘制前需转化为本地颜色格式。

Rop2Code：二元光栅操作码。

返回值：

true：成功；false：失败。

说明：

在 screen 中画一条任意直线，不含端点，如硬件加速不支持在 screen 上画线，driver 可以简化，直接返回 false 即可。有 frame buffer 的情况下，正常显示 gui 不会调用这个函数，如果窗口 direct_screen==true，则可能调用本函数，本函数返回 false 的话，会进一步调用 set_pixel_screen 函数。由于不确定本地显示器是否有 frame_buffer，镜像显示器 driver 必须实现本函数，不能简单返回 false。color 的格式是 CN_SYS_PF_ERGB8888，因此绘制前需转化为本地颜色格式。

此外，函数 line_screen_ie 也是在屏幕上画直线，但是它画的直线包含端点。

5.3.9. fill_rect_screen：屏幕上填充矩形

```
bool_t (*fill_rect_screen)(struct tagRectangle *Target,
                           struct tagRectangle *Focus,
                           u32 Color0,u32 Color1,u32 Mode);
```

参数：

Target：目标矩形区域。

Focus：聚焦矩形区域。

Color0：渐变颜色 0，颜色格式为真彩色。

Color1：渐变颜色 1，颜色格式为真彩色。

Mode：渐变填充模式。

返回值：

true：成功；false：失败。

说明：

screen 中矩形填充，如硬件加速不支持在 screen 上矩形填充，driver 可以简化，直接返回 false 即可。有 frame buffer 的情况下，正常显示 gui 不会调用这个函数，如果窗口 direct_screen==true，则可能调用本函数，本函数返回 false 的话，会进一步调用 set_pixel_screen 函数。由于不知道本地显示器的情况，镜像显示器 driver 必须实现本函数，不能简单返回 false。color 的格式是 CN_SYS_PF_ERGB8888。Target 与 Focus 的关系如图 5-1 所示。

5.3.10. bm_to_screen：屏幕上画位图

```
bool_t (*bm_to_screen)(struct tagRectangle *DstRect,
                       struct tagRectBitmap *SrcBitmap,s32 xSrc,s32 ySrc);
```

参数：

DstRect：目标矩形区域。

SrcBitmap：源位图。

xSrc：源位图中显示的区域左上角横坐标，相对于源位图。

ySrc：源位图中显示的区域左上角纵坐标，相对于源位图。

返回值：

true：成功；false：失败。

说明：

从内存缓冲区到 screen 位块传送，只支持块拷贝，不支持 rop 操作。如硬件加速不支持在 screen 上绘制位图，driver 可以简化，直接返回 false 即可。有 frame buffer 的情况下，__GK_OutputRedraw 中会调用这个函数。如果窗口 direct_screen==true，也可能调用本函数。本函数返回 false 的话，gui kernel 会进一步调用 set_pixel_screen 函数。即使硬件加速支持，但如果不支持具体的 src_bitmap->PixelFormat，也可返回 false。由于不知道本地显示器的情况，镜像显示器 driver 必须实现本函数，不能简单返回 false。

5.3.11. get_pixel_screen：屏幕上读像素

```
u32 (*get_pixel_screen)(s32 x,s32 y);
```

参数：

x：目标像素的横坐标，相对于屏幕。

y：目标像素的纵坐标，相对于屏幕。

返回值：

像素值，转化为真彩色格式。

说明：

从 screen 中取一像素，并转换成 CN_SYS_PF_ERGB8888。

5.3.12. get_rect_screen：屏幕上读位图

bool_t (*get_rect_screen)(struct tagRectangle *Rect,struct tagRectBitmap *Dest);

参数：

rect：读取的目标矩形区域。

dest：保存矩形区域读到的位图。

返回值：

true：成功；false：失败。

说明：

把 screen 内矩形区域的内容复制到 bitmap，调用前，先设置好 dest 的 FixelFormat。

5.3.13. disp_ctrl：控制显示器

bool_t (*disp_ctrl)(struct display_rsc *Disp);

参数：

disp：显示器指针。

返回值：

true：成功；false：失败。

说明：

控制显示器，这是由 driver 提供的一个应用程序的入口，该应用程序用于提供一个可视化的方式。设定该显示器所有可以由用户设定的参数，比如分辨率和和色彩参数。函数的功能不做统一规定，驱动程序的文档应该提供函数的使用说明。利用本函数，可以提供类似 windows 中设置显示器属性的功能。

5.4. API 说明

5.4.1. GK_ApiGetPixelFormat：查询显卡格式

u16 GK_ApiGetPixelFormat(struct tagDisplayRsc *Display)

头文件：

gkernel.h

参数：

Display：待查询的显卡

返回值：

显卡格式，也称颜色格式，如 CN_SYS_PF_RGB565。

说明：

查询显卡使用的颜色格式，画位图时，如果使用跟显卡相同的颜色格式，将获得最优性能。

5.4.2. GK_InstallDisplay：安装显示器

bool_t GK_InstallDisplay(struct tagDisplayRsc *Display, char *Name)

头文件：

gk_display.h

参数：

Display：待安装的显示器。

Name：显示器名称。

返回值：

true：成功；false：失败。

说明：

把一台新显示器登记到显示器资源队列中。

5.4.3. GK_InstallDisplayMirror：安装镜像显示器

bool_t GK_InstallDisplayMirror(struct tagDisplayRsc *BaseDisplay,
struct tagDisplayRsc *MirrorDisplay,char *Name)

头文件：

gk_display.h

参数：

BaseDisplay：本地显示器。

MirrorDisplay：待安装的镜像显示器。

Name：显示器名称。

返回值：

true：成功；false：失败。

说明：

把一台镜像显示器登记到本地显示器。

5.4.4. GK_SetDefaultDisplay：设置默认显示器

bool_t GK_SetDefaultDisplay(char *Name)

头文件：

gk_display.h

参数：

Name：显示器名称。

返回值：

true：成功；false：失败。

说明：

设置默认显示器。

5.4.5. GK_GetRootWin：取显示器的默认设置

struct tagGkWinRsc *GK_GetRootWin(struct tagDisplayRsc *Display)

头文件：

gk_display.h

参数：

Display：显示器指针。

返回值：

显示器桌面指针。

说明：

取一个显示器的默认显示设置，实际上就是桌面窗口的资源节点。

5.4.6. GK_SwitchFrameBuffer：切换帧缓冲

bool_t GK_SwitchFrameBuffer(struct tagDisplayRsc *Display,
struct tagRectBitmap *FrameBuf)

头文件：

gk_display.h

参数：

Display：显示器指针。

FrameBuf：帧缓冲打针。

返回值：

true：成功；false：失败。

说明：

当一个显示器有多个Frame Buffer时，用本函数切换当前Frame Buffer。

5.4.7. GK_CreateFrameBuffer：创建帧缓冲

struct tagRectBitmap *GK_CreateFrameBuffer(struct tagDisplayRsc *Display)

头文件：

gk_display.h

参数：

Display：显示器指针。

返回值：

缓冲区指针。

说明：

为某显示器创建一个 Frame Buffer。

6. 程序示例

6.1. 程序调用步骤

GK 初始化步骤流程如下：

1. 初始化 GK，即内核初始化，调用初始化函数 GK_ModuleInit；
2. 显示器安装，将当前显示器安装到图形系统。
3. 安装字符集，创建桌面和窗口，绘图。

6.2. 参考代码

下面以天嵌开发板 TQ2440 举例，说明 GK 图形化编程。

GK 图形内核的初始化对用户而言，相对简单，大部分工作已经由内核完成，用户只需调用函数 GK_ModuleInit 即可。如所示，初始化内核后，调用函数 module_init_lcd 实现安装名为“ lcd_2440 ”的显示器，返回显示器指针，然后创建了显示器的桌面，并安装字符集。如代码 6-1 所示。

代码 6-1 GK 初始化

```
    GK_ModuleInit(0);
    lcd_2440 = (struct tagDisplayRsc*)module_init_lcd((ptu32_t)"lcd_2440");

    GK_ApiCreateDesktop(lcd_2440,&desktop,0,0,
                        CN_COLOR_RED+CN_COLOR_GREEN,CN_WINBUF_BUF,0,0);

    Gk_Gb2312EncodeModuleInit(0);
```

函数 module_init_lcd 完成的功能是初始化 LCD 硬件，并调用函数 GK_InstallDisplay 实现显示器安装。硬件初始化和显示器安装代码如代码 6-2 所示。

代码 6-2 显示器初始化

```
ptu32_t module_init_lcd(ptu32_t para)
{
    static struct tagGkWinRsc frame_win;
    static struct tagRectBitmap frame_bm;

    __lcd_hard_init();
    __lcd_power_enable(0,1);
    __lcd_envid_of(1);

    frame_bm.PixelFormat = cn_lcd_pf;
    frame_bm.width = cn_lcd_xsize;
    frame_bm.height = cn_lcd_ysize;
    frame_bm.linebytes = cn_lcd_xsize*2;
    frame_bm.bm_bits = (u8*)pg_frame_buffer;

    frame_win.wm_bitmap =&frame_bm;
    tg_lcd_display.frame_buffer = &frame_win;
    tg_lcd_display.xmm = 0;
    tg_lcd_display.ymm = 0;
    tg_lcd_display.width = cn_lcd_xsize;
    tg_lcd_display.height = cn_lcd_ysize;
    tg_lcd_display.pixel_format = CN_SYS_PF_RGB565;
    tg_lcd_display.reset_clip = false;
    tg_lcd_display.framebuf_direct = false;
    //无须初始化frame_buffer和desktop，z_topmost三个成员
    tg_lcd_display.draw.set_pixel_bm = __lcd_set_pixel_bm;
    tg_lcd_display.draw.fill_rect_bm = __lcd_fill_rect_bm;
    tg_lcd_display.draw.get_pixel_bm = __lcd_get_pixel_bm;
    tg_lcd_display.draw.line_bm = __lcd_line_bm;
    tg_lcd_display.draw.line_bm_ie = __lcd_line_bm_ie;
    tg_lcd_display.draw.blt_bm_to_bm = __lcd_blt_bm_to_bm;
    tg_lcd_display.draw.get_rect_bm = __lcd_get_rect_bm;
    tg_lcd_display.draw.set_pixel_screen = __lcd_set_pixel_screen;
    tg_lcd_display.draw.line_screen = __lcd_line_screen;
    tg_lcd_display.draw.line_screen_ie = __lcd_line_screen_ie;
    tg_lcd_display.draw.fill_rect_screen = __lcd_fill_rect_screen;
    tg_lcd_display.draw.bm_to_screen = __lcd_bm_to_screen;
    tg_lcd_display.draw.get_pixel_screen = __lcd_get_pixel_screen;
    tg_lcd_display.draw.get_rect_screen = __lcd_get_rect_screen;

    tg_lcd_display.DisplayHeap =  M_FindHeap("display");
    tg_lcd_display.disp_ctrl = __lcd_disp_ctrl;

    if(GK_InstallDisplay(&tg_lcd_display,(char*)para))
        return (ptu32_t)&tg_lcd_display;
    else
        return (ptu32_t)NULL;
```

```
}
```

完成了 GK 初始化和显示器安装后，创建了第一个桌面后，便就可以实现绘图操作。如代码 6-1 所示，实现在桌面上创建窗口，并输出“ Hello World!”文本到新创建的窗口。

代码 6-3 GK 应用示例

```
struct tagGkWinRsc desktop,win1;

desk = GK_ApiGetDesktop(“lcd_2440”);
if(GK_ ApiCreateGkwin (desktop,win1,20,20,200,200,CN_COLOR_RED,
                          CN_WINBUF_BUF,”WIN1”,CN_SYS_PF_DISPLAY,0,0,0);
{
    GK_ApiDrawText(win1,40,80,”Hello World!”,13,CN_COLOR_BLACK,0,0);
}
```