第9章 颜色

μC/GUI 支持黑/白，灰度（不同亮度的单色）及彩色显示屏。同一个用户程序可以用于不同的显示屏，只需要改变 LCD 配置。色彩管理设法为应该显示的任何色彩找到最相近的匹配。

逻辑颜色：在应用中处理的颜色。一种逻辑颜色总是定义为一个 RGB 数值，这是一个 24 位的数值，其中每个基色 8 位，如：0xBBGGRR。因此，白色应该为 0xFFFFFF，黑色应该为 0x000000，大红为 0xFF0000。

物理颜色：显示屏实际显示的颜色。与逻辑颜色一样，同样定义为一个 24 位的 RGB 数值。在在实际运行的时候，逻辑颜色映射到物理颜色。

对于显示色彩较少的显示屏（例如单色显示屏或 8/16 色 LCD），μC/GUI 通过一个优化版本的“最小平方偏移搜索”对它们进行转换。它对显示的颜色（逻辑颜色）及 LCD 实际能显示（物理颜色）的所有有效颜色进行比较，然后使用 LCD 度量认为最接近的颜色。

9.1 预定义颜色

除自定义颜色之外，在μC/GUI 中预先定义了一些标准的颜色，如下表所示：

定义	颜色	说明
GUI_BLACK	黑	0x000000
GUI_BLUE	蓝	0xFF0000
GUI_GREEN	绿	0x00FF00
GUI_CYAN	青	0xFFFF00
GUI_RED	红	0x0000FF
GUI_MAGENTA	洋红	0x8B008B
GUI_BROW N	褐	0x2A2AA5
GUI_DARKGRAY	深灰	0x404040
GUI_GRAY	灰	0x808080
GUI_LIGHTGRAY	浅灰	0xD3D3D3
GUI_LIGHTBLUE	淡蓝	0xFF8080
GUI_LIGHTGREEN	淡绿	0x80FF80
GUI_LIGHTCYAN	淡青	0x80FFFF
GUI_LIGHTRED	淡红	0x8080FF
GUI_LIGHTMAGENTA	淡洋红	0xFF80FF
GUI_YELLOW	黄	0x00FFFF
GUI_WHITE	白	0xFFFFFF

范例

```
/* 将背景色设为洋红 */
GUI_SetBkColor(GUI_MAGENTA);
GUI_Clear();
```

9.2 色彩条测试程序

下面的色彩条程序用于显示如下所示的 13 种颜色条：

黑 -> 红，白 -> 红，黑 -> 绿，白 -> 绿，黑 -> 蓝，白 -> 蓝，黑 -> 白，黑 -> 黄，白 -> 黄，黑 -> 青，白 ->青，黑 -> 洋红 及 白 -> 洋红。

这个小程序可以以任何颜色格式用于所有显示屏。当然，结果依赖于显示屏能够显示的颜色。该程序要求一个尺寸为 320*240 的显示屏以显示所有颜色。该程序用于证明对于显示屏不同颜色设定的效果。它同时也可能用作一个校验显示屏功能的测试程序，检查有效的颜

色及灰度，同时纠正颜色转换。屏幕截图来自 Windows 仿真器，如果你的设置和硬件是适当的，它看起来和你的显示屏实际输出正确的吻合。该程序在随μC/GUI 一道提供的范例中的名字为：COLOR_ShowColorBar.c。

```
/*----------------------------------------------------------------------
文件:       COLOR _ShowColorBar.c
目的:       绘制一个色彩条的例子
----------------------------------------------------------------------*/

#include "GUI.H"

/*********************************************************************
*                          绘制 13 种颜色的色彩条                    *
*********************************************************************/

void ShowColorBar(void)
{
    int x0 = 60, y0 = 40, yStep = 15, i;
    int NumColors = LCD_GetDevCap(LCD_DEVCAP_NUMCOLORS);
    int xsize = LCD_GetDevCap(LCD_DEVCAP_XSIZE) - x0;
    GUI_SetFont(&GUI_Font13HB_1);
    GUI_DispStringHCenterAt("μC/GUI-sample:Show color bars", 160, 0);
    GUI_SetFont(&GUI_Font8x8);
    GUI_SetColor(GUI_WHITE);
    GUI_SetBkColor(GUI_BLACK);
    #if(LCD_FIXEDPALETTE)
        GUI_DispString("Fixed palette: ");
        GUI_DispDecMin(LCD_FIXEDPALETTE);
    #endif
    GUI_DispStringAt("Red", 0, y0 + yStep);
    GUI_DispStringAt("Green", 0, y0 + 3 * yStep);
    GUI_DispStringAt("Blue", 0, y0 + 5 * yStep);
    GUI_DispStringAt("Grey", 0, y0 + 6 * yStep);
    GUI_DispStringAt("Yellow", 0, y0 + 8 * yStep);
    GUI_DispStringAt("Cyan", 0, y0 + 10 * yStep);
    GUI_DispStringAt("Magenta", 0, y0 + 12 * yStep);
    for(i=0; i < xsize; i++)
    {
```

```
    U16 cs =(255 *(U32)i) / xsize;
    U16 x = x0 + i; ;
/* 红色 */
    GUI_SetColor(cs);
    GUI_DrawVLine(x, y0, y0 +yStep - 1);
    GUI_SetColor(0xff +(255 - cs) * 0x10100L);
    GUI_DrawVLine(x, y0 +yStep, y0 + 2 * yStep - 1);
/* 绿色 */
    GUI_SetColor(cs<<8);
    GUI_DrawVLine(x, y0 + 2 * yStep, y0 + 3 * yStep - 1);
    GUI_SetColor(0xff00 +(255 - cs) * 0x10001L);
    GUI_DrawVLine(x, y0 + 3 * yStep, y0 + 4 * yStep - 1);
/* 蓝色 * /
    GUI_SetColor(cs * 0x10000L);
    GUI_DrawVLine(x, y0 + 4 * yStep, y0 + 5 * yStep - 1);
    GUI_SetColor(0xff0000 +(255 - cs) * 0x101L);
    GUI_DrawVLine(x, y0 + 5 * yStep, y0 + 6 * yStep - 1);
/* 灰色 */
    GUI_SetColor((U32)cs * 0x10101L);
    GUI_DrawVLine(x, y0 + 6 * yStep, y0 + 7 * yStep - 1);
/* 黄色 */
    GUI_SetColor(cs * 0x101);
    GUI_DrawVLine(x, y0 + 7 * yStep, y0 + 8 * yStep - 1);
    GUI_SetColor(0xffff +(255 - cs) * 0x10000L);
    GUI_DrawVLine(x, y0 + 8 * yStep, y0 + 9 * yStep - 1);
/* 青色 */
    GUI_SetColor(cs * 0x10100L);
    GUI_DrawVLine(x, y0 + 9 * yStep, y0 + 10 * yStep - 1);
    GUI_SetColor(0xffff00 +(255 - cs) * 0x1L);
    GUI_DrawVLine(x, y0 + 10 * yStep, y0 + 11 * yStep - 1);
/* 洋红 */
    GUI_SetColor(cs * 0x10001);
    GUI_DrawVLine(x, y0 + 11 * yStep, y0 + 12 * yStep - 1);
    GUI_SetColor(0xff00ff +(255 - cs) * 0x100L);
    GUI_DrawVLine(x, y0 + 12 * yStep, y0 + 13 * yStep - 1);
}
```

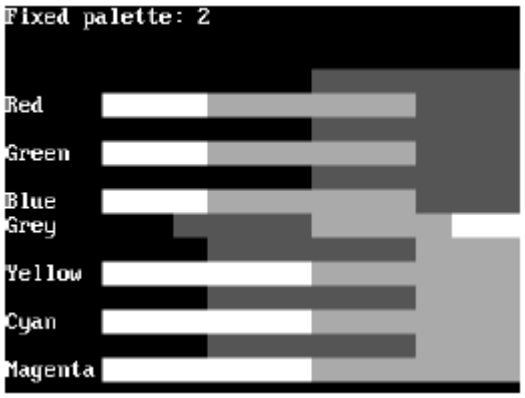
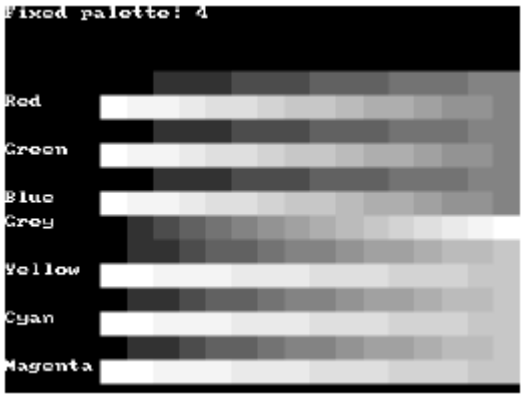
```
/*********************************************************************
*                                主函数                               *
*********************************************************************/

void main(void)
{
    GUI_Init() ;
    ShowColorBar();
    while(1)
        GUI_Delay(100);
}
```

9.3 固定的调色板模式

下表列出了有效的固定调色板颜色模式及必须的“#define”（需要在文件 LCDConf.h 中定义以取得这些模式）。

颜色模式	有效的颜色数	LCD_FIXEDPALETTE	LCD_SWAP_RB
1	2 （黑和白）	1	0
2	4 （灰度）	2	0
4	16 （灰度）	4	0
111	8	111	0
222	64	222	0
233	256	233	0
-233	256	233	1
323	256	323	0
-323	256	323	1
332	256	332	0
-332	256	332	1
444	4096	444	0
555	32768	555	0
-555	32768	555	1
565	65536	565	0
-565	65536	565	1
8666	232	8666	0

详细的描述如下：

1 模式：1 bpp （黑和白） 该模式必须用于每像素 1 位的单色显示屏。 有效颜色数量：2	
2 模式：2 bpp （4 级灰度） 该模式必须用于每像素 2 位的单色显示屏。 有效颜色数量：2×2 = 4.	
4 模式：4 bpp（16 级灰度） 该模式必须用于每像素 4 位的单色显示屏。 有效颜色数量：2×2×2×2＝16	
111 模式：3 bpp （每种基色 2 级） 如果基本的 8 种颜色够用的话可以选择这种模式。如果你的硬件只支持每像素每基色 1 位，或者你没有足够的视频内存用于更高的色彩浓度采用这种模式。 有效颜色数量：2×2×2＝8	

222 模式：6 bpp （每种基色 4 级）

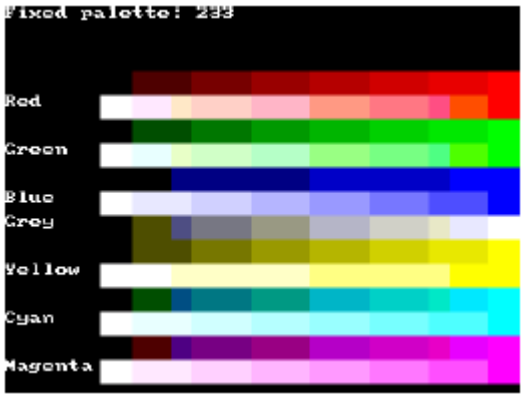
如果你的硬件没有一个调色板用于每种单独的颜色，该模式是一种很好的选择。每像素的每种基色保留 2 位；通常一个像素保存为一个字节。

有效颜色数量：4×4×4＝64

233 模式：2 位蓝色，3 位绿色， 3 位红色

该模式支持 256 种颜色。3 位用于颜色的红色和绿色部分，2 位用于蓝色部分。如图所示，结果是绿色和红色为 8 级，而蓝色为 4 级。有效颜色数量：4×8×8＝256

基色顺序：BBGGGRRR



-233 模式：2 位红色，3 位绿色， 3 位蓝色，红蓝互换

该模式支持 256 种颜色。3 位用于颜色的绿色和蓝色部分，2 位用于红色部分。有效的颜色与 332 模式是一样的。结果绿色和蓝色为 8 级，而红色为 4 级。

有效颜色数量：4×8×8＝256

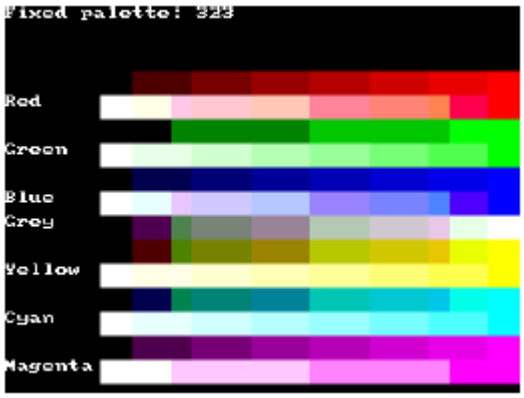
基色顺序：RRGGGBBB

323 模式：3 位蓝色，2 位绿色，3 位红色

该模式支持 256 种颜色。3 位用于颜色的红色和蓝色部分，2 位用于绿色部分。如图所示，结果是蓝色和红色为 8 级，而绿色为 4 级。

有效颜色数量：8×4×8＝256

基色顺序：BBBGGRRR



-323 模式：3 位红色，2 位绿色，3 位蓝色，红蓝互换

该模式支持 256 种颜色。3 位用于颜色的红色和蓝色部分，2 位用于绿色部分。有效的颜色与 323 模式是一样的。结果红色和蓝色为 8 级，而绿色为 4 级。

有效颜色数量：8×4×8＝256

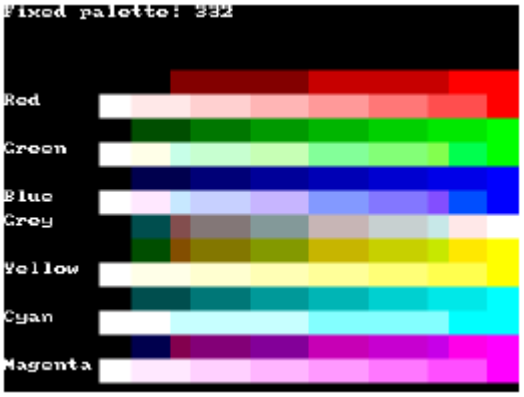
基色顺序：RRRGGBBB

332 模式： 3 位蓝色，3 位绿色，2 位红色

该模式支持 256 种颜色。3 位用于颜色的蓝色和绿色部分，2 位用于红色部分。如图所示，结果是蓝色和绿色为 8 级，而红色为 4 级。

有效颜色数量： 8×8×4＝256

基色顺序：BBBGGGRR



-332 模式：3 位红色，3 位绿色，2 位蓝色，红蓝互换

该模式支持 256 种颜色。3 位用于颜色的红色和绿色部分，2 位用于蓝色部分。有效的颜色与 233 模式是一样的。结果红色和绿色为 8 级，而蓝色为 4 级。

有效颜色数量：8×8×2＝256

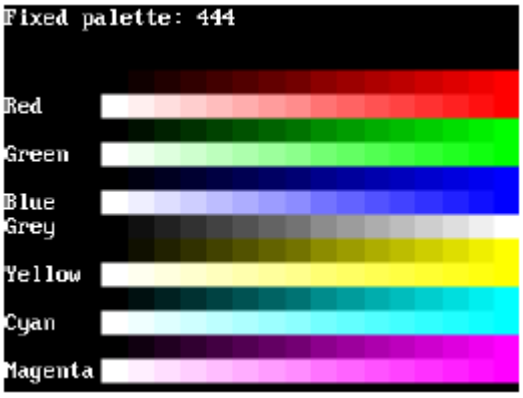
基色顺序：RRRGGGBB

444 模式：4 位红色，4 位绿色，4 位蓝色

红，绿，蓝每部分平均分配 4 位。

有效颜色数量：16×16×16＝4096

基色顺序：BBBBGGGGRRRR

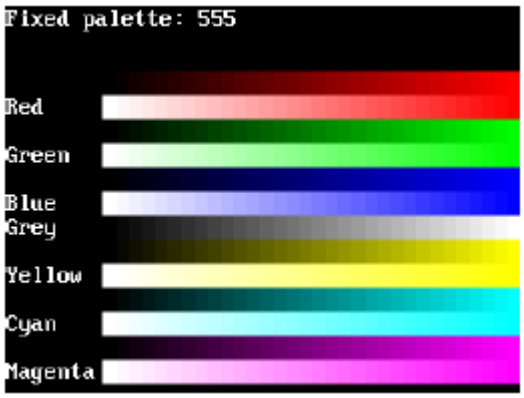


555 模式：5 位红色，5 位绿色，5 位蓝色

使用该模式需要一个支持15bpp的RGB颜色深度的LCD控制器（诸如SED1356或SED13806）。红，绿，蓝每部分平均分配5位。

有效颜色数量：32×32×32＝32768

基色顺序：BBBBBGGGGGRRRRR



-555 模式：5 位蓝色，5 位绿色，5 位红色，红蓝互换

使用该模式需要一个支持 15bpp 的 RGB 颜色深度的 LCD 控制器。红，绿，蓝每部分平均分配 5 位。有效的颜色与 555 模式是一样的。

有效颜色数量： 32×32×32＝32768.

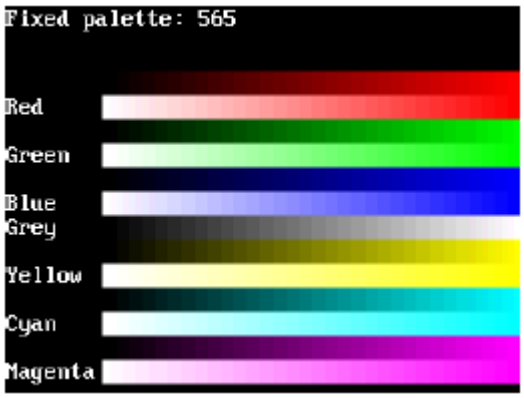
基色顺序：RRRRRGGGGGBBBBB

565 模式： 5 位红色，6 位绿色， 5 位蓝色

使用该模式需要一个支持 16bpp 的 RGB 颜色深度的 LCD 控制器红，绿，蓝每部分平均分配 5 位。一位未使用到。

有效颜色数量： 32×64×32＝65536.

基色顺序：BBBBBGGGGGGRRRRR

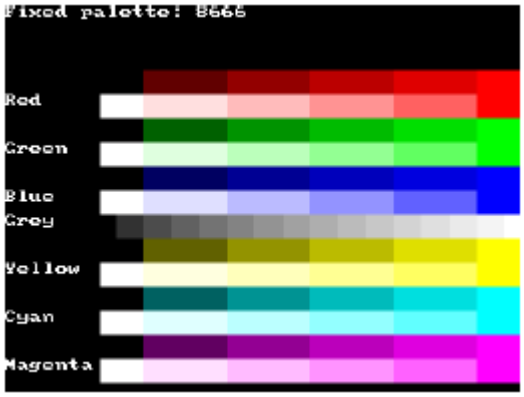


-565 模式： 5 位蓝色，6 位绿色，5 位红色，红蓝互换

使用该模式需要一个支持 16bpp 的 RGB 颜色深度的 LCD 控制器红，绿，蓝每部分平均分配 5 位。一位未使用到。有效的颜色与 565 模式是一样的。

有效颜色数量： 32×64×32＝65536.

基色顺序：RRRRRGGGGGGBBBBB

8666 模式：8bpp，每种基色 6 级＋ 16 级灰度 该模式多数用于一个可编程的颜色查询表（LUT），支持一个调色板总共 256 种可能的颜色。屏幕截图给出了一种有效颜色的意见，该模式包括一般目的应用的最好选择。每种基色 6 级有效亮度，附加 16 级灰度。 有效颜色数量： 6×6×6+16＝232	

9.4 定制调色板模式

µC/GUI 能处理一个定制的硬件调色板。定制调色板仅仅列出为硬件所使用的同一序列的所有有效的颜色。意思是无论如何你的 LCD 控制器/显示屏组件都能显示这些颜色。µC/ GUI 能在 PC 仿真器上仿真这种应用，从而正确的在你的目标系统中处理这些颜色。

为了定义一个定制调色板，你应该在配置文件 LCDConf.h 中这样做。

范例

下面例子（LCDConf.h 部分）将定义一个定制调色板，其中包括 4 种颜色，它们全都是灰度色：

```
#define LCD_FIXEDPALETTE 0
#define LCD_PHYSCOLORS 0xffffff, 0xaaaaaa, 0x555555, 0x000000
```

9.5 在运行时修改颜色查询表

每个像素的颜色信息既可以保存为 RGB 模式（每像素保持红、绿、蓝三部分），也可以保存为索引色模式（一个叫颜色索引的单个数字保存每个像素颜色信息）。每种颜色索引都反映一个查询表的条目，或者颜色映射，定义一个指定的 RGB 数值。

如果你的 LCD 控制器让一个颜色查询表（LUT）起作用，它完全可能通过µC/GUI 在初始化阶段（GUI_Init() -> LCD_Init() -> LCD_InitLUT() -> LCD_L0_SetLUTEntry()）进行初始化。然而，它可能需要在运行时修改 LUT（因为不同的理由）。一些可能的理由包括：

- 为了补偿显示屏问题（非线性）或进行伽马修正而进行颜色修正
- 显示屏的反相显示

- 需要使用比硬件能显示颜色（在一个时间段）更多的颜色（在不同的时间）

如果你仅仅在运行的时候修改 LUT，颜色转换函数将不会意识到这种改变，依然假定 LUT 是初始化时的原始状态。

使用不同的颜色

颜色表的缺省内容在对配置文件 GUIConf.h（LCD_PHYS_COLORS）进行编译时被定义。为了尽可能减少 RAM 的浪费，这个数据通常声明为常量，因此它能保存在 ROM 中。为了能修改它，需要将它调入 RAM 中。这可能通过激活配置开关 LCD_LUT_IN_RAM 来完成。如果这激活了，API 函数 GUI_SetLUTColor()会有效，就能用于在同一时间修改颜色表和 LUT 的内容。

调用 LCD_InitLUT()函数将会恢复原始(默认)设置。

9.6 颜色API

下表列出了与颜色处理相关的函数，在各自的类型中按字母顺序进行排列。函数的详细描述后面列出。

函 数	说 明
基本颜色函数	
GUI_GetBkColor()	返回当前背景颜色
GUI_GetBkColorIndex()	返回当前背景颜色的索引
GUI_GetColor()	返回当前前景颜色
GUI_GetColorIndex()	返回当前前景颜色的索引
GUI_SetBkColor()	设置当前背景颜色
GUI_SetBkColorIndex()	设置当前背景颜色索引
GUI_SetColor()	设置当前前景颜色
GUI_SetColorIndex()	设置当前前景颜色索引
索引色及全彩色转换	
GUI_Color2Index()	将全彩色转换成索引色
GUI_Index2Color()	将索引色转换成全彩色.
查询表（LUT）类	
GUI_InitLUT()	初始化 LUT（硬件）
GUI_SetLUTColor()	设置一种索引色的颜色（硬件及颜色表）
GUI_SetLUTEntry()	向 LUT 写入一个数值（硬件）

9.7 基本颜色函数

GUI_GetBkColor()

描述

返回当前背景颜色。

函数原型

```
GUI COLOR GUI GetBkColor(void);
```

返回值

当前的背景颜色色。

GUI_GetBkColorIndex()

描述

返回当前背景颜色的索引

函数原型

```
int GUI_GetBkColorIndex(void);
```

返回值

当前背景颜色的索引

GUI_GetColor()

描述

返回当前前景颜色。

函数原型

```
GUI COLOR GUI GetColor(void);
```

返回值

当前前景颜色。

GUI_GetColorIndex()

描述

返回当前前景色索引

函数原型

```
int GUI_GetColorIndex(void);
```

返回值

当前前景色索引

GUI_SetBkColor()

描述

设置当前背景颜色。

函数原型

```
GUI_COLOR GUI_SetBkColor(GUI_COLOR Color);
```

参　数	含　意
Color	背景颜色，24 位 RGB 数值。

返回值

选择的背景颜色

GUI_SetBkColorIndex()

描述

设置当前背景色的索引值

函数原型

```
int GUI_SetBkColorIndex(int Index);
```

参 数	含 意
Index	要使用的颜色的索引值。

返回值

所选择背景颜色的索引值

GUI_SetColor()

描述

设置当前前景色

函数原型

```
void GUI_SetColor(GUI_COLOR Color);
```

参 数	含 意
Color	前景颜色，24 位 RGB 数值。

返回值

所选择的前景颜色

GUI_SetColorIndex()

描述

设置当前前景色的索引值

函数原型

```
void GUI_SetColorIndex(int Index);
```

参 数	含 意
Index	要使用的颜色的索引值。

返回值

所选择前景颜色的索引值

9.8 索引和彩色转换

GUI_Color2Index()

描述

返回一种指定的 RGB 颜色数值的索引值

函数原型

```
int GUI_Color2Index(GUI_COLOR Color)
```

参　数	含　意
Color	要转换的颜色的 RGB 值。

返回值

颜色的索引值

GUI_Index2Color()

描述

返回一种指定的索引颜色的 RGB 颜色值

函数原型

```
int GUI_Index2Color(int Index)
```

参　数	含　意
Index	要转换的颜色的索引值

返回值

RGB 颜色值

9.9 查询表（LUT）类

这些函数是可选择的，而且只能在 LCD 控制器硬件支持的情况下工作。要求一个带 LUT 硬件的 LCD 控制器。请参考你所使用的 LCD 控制器的手册以获得关于 LUT 的更多信息。

GUI_InitLUT()

描述

初始化 LCD 控制器的查询表

函数原型

```
void LCD_InitLUT(void) ;
```

附加信息

该函数需要激活查询表（通过宏 LCD_INITCONTROLLER）才有效果

GUI_SetLUTColor()

描述

修改颜色表和 LCD 控制器的 LUT 中的单个条目

函数原型

```
void GUI_SetLUTColor(U8 Pos, GUI_COLOR Color);
```

参　数	含　意
Pos	查询表中的位置。应该小于颜色数（例如，对于 2bpp 为 0～3，4bpp 为 0～15，8bpp 为 0～255）
Color	24 位 RGB 数值

附加信息

最接近的可能数值将用于 LUT。如果一个颜色 LUT 已经初始化，所有三个部分基色都会用到。在单色模式下，用到绿色部分，但是仍然推荐（为了更好的理解程序代码）将三种基色设置为同一个数值（例如 0x555555 或 0xa0a0a0）。

该函数需要激活查询表（通过宏 LCD_INITCONTROLLER）才有效果。该函数总是有效的，但是只有在下面两个条件下才有效果：

- 如果 LUT 已经使用
- 颜色表位次于 RAM 中（LCD_PHYSCOLORS_IN_RAM）

GUI_SetLUTEntry()

描述

修改 LCD 控制器的 LUT 的中的单个条目

函数原型

```
void GUI_SetLUTEntry(U8 Pos, GUI_COLOR Color);
```

参　数	含　意
Pos	查询表中的位置。应该小于颜色数（例如，对于 2bpp 为 0～3，4bpp 为 0～15，8bpp 为 0～255）
Color	24 位 RGB 数值

附加信息

最接近的可能数值将用于 LUT。如果一个颜色 LUT 已经初始化，所有三个部分基色都会用到。在单色模式下，用到绿色部分，但是仍然推荐（为了更好的理解程序代码）将三种基色设置为同一个数值（例如 0X555555 或 0Xa0a0a0）。

该函数需要激活查询表（通过宏 LCD_INITCONTROLLER）才有效果。该函数常常用于确保实际显示的颜色与逻辑颜色相匹配（线性化）。

范例

```
// 线性化一个 4 级灰度 LCD 的调色板
GUI_SetLUTEntry(0, 0x000000);
GUI_SetLUTEntry(1, 0x777777);
// GUI_SetLUTEntry(2, 0xbbbbbb)线性化结果是 555555;
// GUI_SetLUTEntry(3, 0xffffff)线性化结果是 aaaaaa;
```