# 第6章 2-D图形库

µC/GUI包括有一个完整的2-D图形库，在大多数场下应用是足够了。µC/GUI提供的函数既可以与裁剪区一道使用也可以脱离裁剪区使用（参考第12章“视窗管理器”），这些函数基于快速及有效率的算法建立。

目前，只有绘制圆弧函数要求浮点运算支持。

# 6.1 API参考：图形

下表列出了与图形处理相关的函数，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 |
|---|---|
| **绘图模式** | |
| GUI_SetDrawMode() | 设置绘图模式。 |
| **基本绘图函数** | |
| GUI_ClearRect() | 使用背景颜色填充一个矩形区域。 |
| GUI_DrawPixel() | 绘一个单像素点。 |
| GUI_DrawPoint() | 绘一个点。 |
| GUI_FillRect() | 绘一个填充的矩形。 |
| GUI_InvertRect() | 反转一个矩形区域。 |
| **绘制位图** | |
| GUI_DrawBitmap() | 绘制一幅位图。 |
| GUI_DrawBitmapExp() | 绘制一幅位图。 |
| GUI_DrawBitmapMag() | 绘制一幅放大的位图。 |
| GUI_DrawStreamedBitmap() | 从一个位图数据流的数据绘制一幅位图。 |
| **绘线** | |
| GUI_DrawHLine() | 绘一根水平线。 |
| GUI_DrawLine() | 绘一根线。 |
| GUI_DrawLineRel() | 从当前坐标到端点绘一根线，该端点由X轴距离及Y轴距离指定。 |
| GUI_DrawLineTo() | 从当前坐标到端点(X,Y)绘一根线。 |
| GUI_DrawPolyLine() | 绘折线。 |
| GUI_DrawVLine() | 绘一根垂直线。 |
| **绘多边形** | |
| GUI_DrawPolygon() | 绘一个多边形。 |
| GUI_EnlargePolygon() | 对一个多边形进行扩边。 |
| GUI_FillPolygon() | 绘一个填充的多边形。 |
| GUI_MagnifyPolygon() | 放大一个多边形。 |
| GUI_RotatePolygon() | 按指定角度旋转一个多边形。 |
| **绘圆** | |
| GUI_DrawCircle() | 绘一个圆。 |
| GUI_FillCircle() | 绘一个填充的圆。 |
| **绘椭圆** | |
| GUI_DrawEllipse() | 绘一个椭圆。 |
| GUI_FillEllipse() | 绘一个填充的椭圆。 |

| 绘圆弧 | |
|---|---|
| GUI_DrawArc() | 绘一个圆弧 |

## 6.2 绘图模式

µC/GUI提供两种绘图模式，NORMAL模式及XOR模式。默认为NORMAL模式，即显示屏的内容被绘图所完全覆盖。在XOR模式，当绘图覆盖在上面时，显示屏的内容反相显示。

### 与GUI_DRAWMODE_XOR有关的限制

- XOR模式通常用于在活动视窗或屏幕中使用两种颜色进行显示的场合。
- 一些µC/GUI的绘图函数并不能正确地工作在这种模式。通常情况下，这模式只是工作于一个像素大小的笔尖尺寸。这意味着在使用类似GUI_DrawLine，GUI_DrawCircle，GUI_DrawRect等等这样的函数之前，你必须确定在XOR模式下，笔尖尺寸已经设为1。
- 当使用颜色的深度大于1位/像素（bpp）进行位图绘制，该模式无效。
- 当使用诸如GUI_DrawPolyLine这样的函数或多次调用GUI_DrawLineTo函数，转角点会反相两次。结果是这些像素保持背景颜色。

### GUI_SetDrawMode

#### 描述

选择指定的绘图模式

#### 函数原型

```
GUI_DRAWMODE GUI_SetDrawMode(GUI_DRAWMODE mode);
```

| 参　数 | 含　意 |
|---|---|
| mode | 设置的绘图模式。可以是任意设置绘图模式的函数的返回值或是下表中的任一个。 |

参数mode允许的数值

| | |
|---|---|
| GUI_DRAWMODE_NORMAL | 默认：绘点，线，区域，位图 |
| GUI_DRAWMODE_XOR | 当在屏幕上另一个物体上用颜色覆盖时对点，线，区域进行反相显示 |

#### 返回值

所选择的绘图模式

**附加信息**

作为设置绘图模式的附加功能，该函数也可以用于恢复原先被修改的绘图模式。

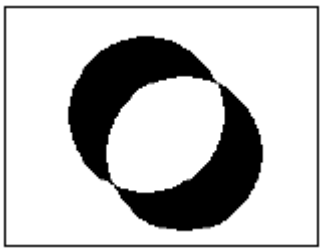
如果使用颜色，一个反相的像素由下式算出：

新像素颜色 = 颜色的数值 - 实际像素颜色 - 1

**范例**

```
// 显示两个圆，其中第二个以XOR模式与第一个结合
GUI_Clear();
GUI_SetDrawMode(GUI_DRAWMODE_NORMAL);
GUI_FillCircle(120, 64, 40);
GUI_SetDrawMode(GUI_DRAWMODE_XOR);
GUI_FillCircle(140, 84, 40);
```

**上面范例程序运行结果的屏幕截图**



# 6.3 基本绘图函数

基本绘图函数允许在显示屏上的任何位置进行单独的点，水平和垂直线段和形状的绘制。使用任何有效的绘图模式。因为这些函数在大多数应用中被频繁调用，因此它们已经被尽量地优化以获得尽可能快的速度。例如，水平和垂直线段绘制函数不需要使用单个点绘制函数。

GUI_ClearRect

**描述**

在当前视窗的指定位置通过向一个矩形区域填充背景色来清除它。

**函数原型**

```
void GUI_ClearRect(int x0, int y0, int x1, int y1);
```

| 参　数 | 含　意 |
|---|---|
| x0 | 左上角X坐标 |
| y0 | 左上角Y坐标 |
| x1 | 右下角X坐标 |
| y0 | 右下角Y坐标 |

**相关主题**

GUI_InvertRect，GUI_FillRect

## GUI_DrawPixel

**描述**

在当前视窗的指定坐标绘一个像素点。

**函数原型**

```
void GUI_DrawPixel(int x, int y);
```

| 参　数 | 含　意 |
|---|---|
| x | 像素点的X坐标 |
| y | 像素点的Y坐标 |

**相关主题**

GUI_DrawPoint

## GUI_DrawPoint

**描述**

在当前视窗使用当前尺寸笔尖绘一个点。

**函数原型**

```
void GUI_DrawPoint(int x, int y);
```

| 参　数 | 含　意 |
|---|---|
| x | 点的X坐标 |
| y | 点的Y坐标 |

**相关主题**

GUI_DrawPixel

## GUI_FillRect

**描述**

在当前视窗指定的位置绘一个矩形填充区域。

**函数原型**

```
void GUI_FillRect(int x0, int y0, int x1, int y1);
```

| 参　数 | 含　意 |
|---|---|
| x0 | 左上角X坐标 |
| y0 | 左上角Y坐标 |
| x1 | 右下角X坐标 |
| y0 | 右下角Y坐标 |

**附加信息**

使用当前的绘图模式，通常表示在矩形内的所有像素都被设置。

**相关主题**

GUI_InvertRect，GUI_ClearRect

## GUI_InvertRect

**描述**

在当前视窗的指定位置绘一反相的矩形区域。

**函数原型**

```
void GUI_InvertRect(int x0, int y0, int x1, int y1);
```

| 参　数 | 含　意 |
|---|---|
| x0 | 左上角X坐标 |
| y0 | 左上角Y坐标 |
| x1 | 右下角X坐标 |
| y0 | 右下角Y坐标 |

**相关主题**

GUI_FillRect，GUI_ClearRect

# 6.4 绘制位图

## GUI_DrawBitmap

**描述**

在当前视窗的指定位置绘一幅位图。

**函数原型**

```
void GUI_DrawBitmap(const GUI_BITMAP*pBM, int x, int y);
```

| 参　数 | 含　意 |
|---|---|
| pBM | 需显示位图的指针 |
| x | 位图在屏幕上位置的左上角X坐标 |
| y | 位图在屏幕上位置的左上角Y坐标 |

**附加信息**

位图数据必须定义为像素×像素。每个像素等同于一位。最高有效位（MSB）定义第一个像素；图片数据以位流进行说明，以第一个字节的MSB作为起始。新的一行总是在一个偶数地址开始，而位图的第N行在地址偏移量n* BytesPerLine处开始。位图可以在客户区中任意一点显示，位图转换器用于产生位图。

**范例**

```
extern const GUI_BITMAP bmMicriumLogo;       /* 声明外部位图 */
void main()
```

```
{
    GUI_Init();
    GUI_DrawBitmap(&bmMicriumLogo,45,20);
}
```

**上面范例程序运行结果的屏幕截图**


## GUI_DrawBitmapExp

### 描述

与GUI_DrawBitmap函数具有相同功能，但是带有扩展参数设置。

### 函数原型

```
void GUI_DrawBitmapExp( int x0, int y0,
                        int XSize, int YSize,
                        int XMul, int YMul,
                        int BitsPerPixel,
                        int BytesPerLine,
                        const U8* pData,
                        const GUI_LOGPALETTE* pPal);
```

| 参　数 | 含　意 |
|---|---|
| x | 位图在屏幕上位置的左上角X坐标 |
| y | 位图在屏幕上位置的左上角Y坐标 |
| Xsize | 水平方向像素的数量，有效范围：1～255 |
| Ysize | 垂直方向像素的数量，有效范围：1～255 |
| XMuL | X轴方向比例因数 |
| YMuL | Y轴方向比例因数 |
| BitsPerPixel | 每像素的位数 |
| BytesPerLine | 图形每行的字节数 |
| pData | 实际图形的指针，该数据定义位图的外表特征 |
| pPal | 指向GUI_LOGPALETTE结构的指针 |

## GUI_DrawBitmapMag

### 描述

该函数使在屏幕上放缩一幅位图成为可能。

### 函数原型

```
void GUI_DrawBitmapMag( const GUI_BITMAP* pBM,
                        int x0, int y0,
                        int XMul, int YMul);
```

| 参　数 | 含　意 |
|---|---|
| pBM | 所显示位图的指针 |
| x0 | 位图在屏幕上位置的左上角X坐标 |
| y0 | 位图在屏幕上位置的左上角Y坐标 |
| XMul | X轴方向比例因数 |
| YMul | Y轴方向比例因数 |

## GUI_DrawStreamedBitmap

### 描述

该函数从一个位图数据流的数据绘制一幅位图。

### 函数原型

```
void GUI_DrawStreamedBitmap (   const GUI_BITMAP_STREAM *pBMH,
                                int x,
                                int y);
```

| 参　数 | 含　意 |
|---|---|
| pBMH | 指向数据流的指针 |
| x | 位图在屏幕上位置的左上角X坐标 |
| y | 位图在屏幕上位置的左上角Y坐标 |

### 附加信息

你可以使用在后面的章节描述的位图转换器建立位图数据流。这些数据流的格式与BMP文件的格式不一样。

# 6.5 绘线

绘图函数使用频率最高的是那些从一个点到另一个点的绘线函数。

## GUI_DrawHLine

### 描述

在当前视窗从一个指定的起点到一个指定的终点，以一个像素厚度画一条水平线。

### 函数原型

```
void GUI_DrawHLine(int y, int x0, int x1);
```

| 参　数 | 含　意 |
|---|---|
| y | Y轴坐标 |
| x0 | 起点的X轴坐标 |
| x1 | 终点的X轴坐标 |

### 附加信息

对于大多数LCD控制器，该函数执行速度非常快，因为多个像素能马上布置好，无需进行计算。很明显在绘水平线方面，该函数执行比GUI_DrawLine函数还要快。

## GUI_DrawLine

### 描述

在当前视窗的指定始点到指定终点绘一条直线。

### 函数原型

```
void GUI_DrawLine(int x0, int y0, int x1, int y1);
```

| 参　数 | 含　意 |
|---|---|
| x0 | X轴开始坐标 |
| y0 | Y轴开始坐标 |
| x1 | X轴结束坐标 |
| y1 | Y轴结束坐标 |

**附加信息**

如果线的一部分是不可见的，因为它不在当前视窗内，或者如果当前视窗的一部分是不可见的，由于裁剪的原因，这些部分将不会绘出。

## GUI_DrawLineRel

**描述**

在当前视窗从当前坐标(X,Y)到一个端点绘一条直线，指定X轴距离和Y轴距离。

**函数原型**

```
void GUI_DrawLineRel(int dx, int dy);
```

| 参　数 | 含　意 |
|---|---|
| dx | 到所绘直线末端X轴方向的距离 |
| dy | 到所绘直线末端Y轴方向的距离 |

## GUI_DrawLineTo

**描述**

在当前视窗从当前坐标(X,Y)到一个端点绘一条直线，指定端点的X轴，Y轴坐标。

**函数原型**

```
void GUI_DrawLineTo(int x, int y);
```

| 参　数 | 含　意 |
|---|---|
| x | 终点的X轴坐标 |
| y | 终点的Y轴坐标 |

## GUI_DrawPolyLine

**描述**

在当前视窗中用直线连接一系列预先确定的点。

**函数原型**

```
void GUI_DrawPolyLine(const GUI_POINT* pPoint, int NumPoints, int x, int y);
```

| 参　数 | 含　意 |
|---|---|
| pPoint | 指向所显示的折线的指针 |
| NumPoints | 点系列中指定点的数量 |
| x | 原点的X轴坐标 |
| y | 原点的Y轴坐标 |

**附加信息**

折线的起点和终点不要求重合。

### GUI_DrawVLine

**描述**

在当前视窗从一个指定的起点到一个指定的终点，以一个像素厚度画一条垂直线。

**函数原型**

```
void GUI_DrawVLine(int x, int y0, int y1);
```

| 参　数 | 含　意 |
|---|---|
| x | X轴坐标 |
| y0 | 起点的Y轴坐标 |
| y1 | 终点的Y轴坐标 |

**附加信息**

如果y1＜y0，则不会有任何显示。

对于大多数LCD控制器，该函数执行速度非常快，因为多个像素能马上布置好，无需进行计算。很明显在绘垂线方面，该函数执行比GUI_DrawLine函数还要快。

## 6.6 绘多边形

在绘矢量符号时绘多边形函数很有用。

## GUI_DrawPolygon

### 描述

在当前视窗中绘一个由一系列点定义的多边形的轮廓。

### 函数原型

```
void GUI_DrawPolygon(const GUI_POINT* pPoint, int NumPoints, int x, int y);
```

| 参　数 | 含　意 |
|---|---|
| pPoint | 显示的多边形的指针 |
| Numpoints | 在点的序列中指定点的数量 |
| x | 原点的X轴坐标 |
| y | 原点的Y轴坐标 |

### 附加信息

所绘的折线通过连接起点和终点而自动闭合。

## GUI_EnlargePolygon

### 描述

通过指定一个以像素为单位的长度，对多边形的所有边进行放大（扩边，与对多边形进行放大的概念不一样）。

### 函数原型

```
void GUI_EnlargePolygon (   GUI_POINT* pDest,
                            const GUI_POINT* pSrc,
                            int NumPoints,
                            int Len);
```

| 参　数 | 含　意 |
|---|---|
| pPoint | 目标多边形的指针 |
| pSrc | 源多边形的指针 |
| Numpoints | 在点的序列中指定点的数量 |
| Len | 对多边形进行放大的以像素为单位的长度 |

**附加信息**

确认目标点的阵列等于或大于源阵列。

**范例**

```
#define countof(Array) (sizeof(Array) / sizeof(Array[0]))
const GUI_POINT aPoints[] = {
    {0, 20},
    {40, 20},
    {20, 0}
};
GUI_POINT aEnlargedPoints[countof(aPoints)];
void Sample(void)
{
    int i;
    GUI_Clear();
    GUI_SetDrawMode(GUI_DM_XOR);
    GUI_FillPolygon(aPoints, countof(aPoints), 140, 110);
    for (i = 1; i < 10; i++)
    {
        GUI_EnlargePolygon( aEnlargedPoints, aPoints,
                            countof(aPoints), i * 5);
        GUI_FillPolygon(aEnlargedPoints, countof(aPoints), 140, 110);
    }
}
```

**上面范例程序运行结果的屏幕截图**

## GUI_FillPolygon

### 描述

在当前视窗中绘一个由一系列点定义的填充多边形。

### 函数原型

```
void GUI_FillPolygon(const GUI_POINT* pPoint, int NumPoints, int x, int y);
```

| 参　数 | 含　意 |
|---|---|
| pPoint | 显示的填充多边形的指针 |
| Numpoints | 在点的序列中指定点的数量 |
| x | 原点的X轴坐标 |
| y | 原点的Y轴坐标 |

### 附加信息

绘制的折线通过连接起点与终点而自动闭合。终点必须不能与多边形的轮廓接触。

## GUI_MagnifyPolygon

### 描述

通过指定一个因数对多边形进行放大。

### 函数原型

```
void GUI_MagnifyPolygon (   GUI_POINT* pDest,
                            const GUI_POINT* pSrc,
                            int NumPoints,
                            int Mag);
```

| 参　数 | 含　意 |
|---|---|
| pDest | 目标多边形的指针 |
| pSrc | 源多边形的指针 |
| Numpoints | 在点的序列中指定点的数量 |
| Mag | 多边形的放大因数 |

### 附加信息

确认目标点的阵列等于或大于源阵列。注意对多边形进行扩边与放大的不同。设置参数Len为1将会对多边形的所有边进行1个像素的扩大，而设置参数Mag为1，则对多边形没有任何影响。

**范例**

```
#define countof(Array) (sizeof(Array)/sizeof(Array[0]))
const GUI_POINT aPoints[] = {
    {0, 20},
    {40, 20},
    {20, 0}
};
GUI_POINT aMagnifiedPoints[countof(aPoints)];
void Sample(void)
{
    int Mag, y = 0, Count = 4;
    GUI_Clear();
    GUI_SetColor(GUI_GREEN);
    for (Mag = 1; Mag <= 4; Mag *= 2, Count /= 2)
    {
        int i, x = 0;
        GUI_MagnifyPolygon( aMagnifiedPoints, aPoints,
                            countof(aPoints), Mag);
        for (i = Count; i > 0; i--, x += 40 * Mag)
        {
            GUI_FillPolygon(aMagnifiedPoints, countof(aPoints), x, y);
        }
        y += 20 * Mag;
    }
}
```

**上面范例程序运行结果的屏幕截图**

## GUI_RotatePolygon

### 描述

按指定角度旋转一个多边形。

### 函数原型

```
void GUI_RotatePolygon( GUI_POINT* pDest,
                        const GUI_POINT* pSrc,
                        int NumPoints,
                        float Angle);
```

| 参　数 | 含　意 |
|---|---|
| pDest | 目标多边形的指针 |
| pSrc | 源多边形的指针 |
| Numpoints | 在点的序列中指定点的数量 |
| Angle | 多边形旋转的角度（以弧度为单位） |

### 附加信息

确认目标点的阵列等于或大于源阵列。

### 范例

下面的范例显示如何画一个多边形。该范例源程序文件是“Samples\Misc\DrawPolygon.c”。

```
/*----------------------------------------------------------------------
文件:        DrawPolygon.c
```

```
目的:       绘制一个多边形
---------------------------------------------------------------------------*/

#include "GUI.H"

/*****************************************************************************
*                                  箭头的点                                  *
*****************************************************************************/

static const GUI_POINT aPointArrow[] = {
    { 0, -5},
    {-40, -35},
    {-10, -25},
    {-10, -85},
    { 10, -85},
    { 10, -25},
    { 40, -35},
};

/*****************************************************************************
*                                  绘制一个多边形                            *
*****************************************************************************/

static void DrawPolygon(void)
{
    int Cnt =0;
    GUI_SetBkColor(GUI_WHITE);
    GUI_Clear();
    GUI_SetFont(&GUI_Font8x16);
    GUI_SetColor(0x0);
    GUI_DispStringAt("Polygons of arbitrary shape", 0, 0);
    GUI_DispStringAt("in any color", 120, 20);
    GUI_SetColor(GUI_BLUE);
/* 画一个填充多边形 */
    GUI_FillPolygon (&aPointArrow[0],7,100,100);
}
```

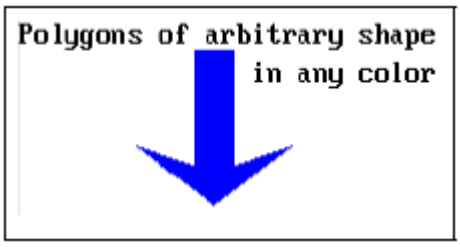
**参数含意**

pDest： 目标多边形的指针。
pSrc： 源多边形的指针
NumPoints： 在一个点的序列中指定点的数量
Angle： 旋转多边形的角度（以弧度为单位）

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

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

**上面范例程序运行结果的屏幕截图**




# 6.7 绘圆

## GUI_DrawCircle

**描述**

在当前视窗指定坐标以指定的尺寸绘制一个圆。

**函数原型**

```
void GUI_DrawCircle(int x0, int y0, int r);
```

| 参 数 | 含 意 |
|---|---|
| x0 | 在客户视窗中圆心的X轴坐标（以像素为单位） |
| y0 | 在客户视窗中圆心的Y轴坐标（以像素为单位） |
| r | 圆的半径（直径的一半）。<br>最小值：0（结果是一个点）<br>最大值：180 |

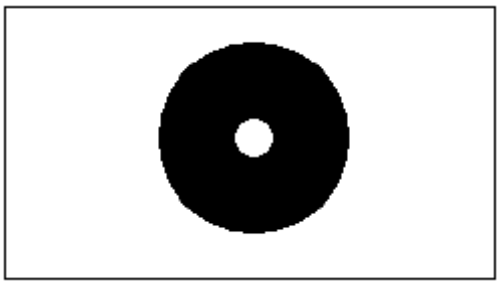
**附加信息**

该函数不能处理超过180的半径，因为那样将使用到导致溢出错误的整数运算。但是，对于大多数嵌入式应用，这不算是一个问题，因为一个直径为360的圆无论如何都会比显示屏要大。

**范例**

```
// 画同心圆
void ShowCircles(void)
{
    int i;
    for (i=10; i<50; i++)
        GUI_DrawCircle(120,60,i);
}
```

**上面范例程序执行结果的屏幕截图**



## GUI_FillCircle

**描述**

在当前视窗指定坐标以指定的尺寸绘制一个填充圆。

**函数原型**

```
void GUI_FillCircle(int x0, int y0, int r);
```

| 参 数 | 含 意 |
|---|---|
| x0 | 在客户视窗中圆心的X轴坐标（以像素为单位） |
| y0 | 在客户视窗中圆心的Y轴坐标（以像素为单位） |
| r | 圆的半径（直径的一半）。<br>最小值：0（结果是一个点）<br>最大值：180 |

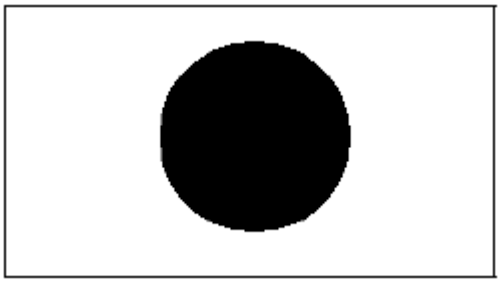
**附加信息**

该函数不能处理超过180的半径。

**范例**

```
GUI_FillCircle(120,60,50);
```

**上面范例程序执行结果的屏幕截图**



# 6.8 绘椭圆

## GUI_DrawEllipse

**描述**

在当前视窗的指定坐标以指定的尺寸绘一个椭圆。

**函数原型**

```
void GUI_DrawEllipse (int x0, int y0, int rx, int ry);
```

| 参　数 | 含　意 |
|---|---|
| x0 | 在客户视窗中圆心的X轴坐标（以像素为单位） |
| y0 | 在客户视窗中圆心的Y轴坐标（以像素为单位） |
| rx | 椭圆的X轴半径（直径的一半）。<br>最小值：0，最大值：180 |
| ry | 椭圆的Y轴半径（直径的一半）。<br>最小值：0，最大值：180 |

**附加信息**

该函数处理的rx/ry参数不能超过180，因为那样将使用到导致溢出错误的整数运算。

**范例**

参考GUI_FillEllipse函数的范例

## GUI_FillEllipse

**描述**

按指定的尺寸绘一个填充椭圆。

**函数原型**

```
void GUI_FillEllipse(int x0, int y0, int rx, int ry);
```

| 参　数 | 含　意 |
|---|---|
| x0 | 在客户视窗中圆心的X轴坐标（以像素为单位） |
| y0 | 在客户视窗中圆心的Y轴坐标（以像素为单位） |
| rx | 椭圆的X轴半径（直径的一半）。<br>最小值：0，最大值：180 |
| ry | 椭圆的Y轴半径（直径的一半）。<br>最小值：0，最大值：180 |

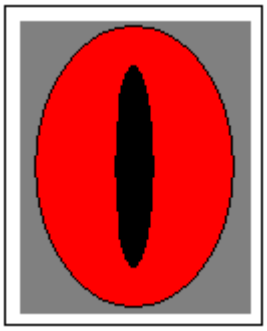
**附加信息**

该函数处理的rx/ry参数不能超过180。

**范例**

```
/* 椭圆范例 */
GUI_SetColor(0xff);
GUI_FillEllipse(100, 180, 50, 70);
GUI_SetColor(0x0);
GUI_DrawEllipse(100, 180, 50, 70);
GUI_SetColor(0x000000);
GUI_FillEllipse(100, 180, 10, 50);
```

**上面范例程序执行结果的屏幕截图**



# 6.9 绘制圆弧

## GUI_DrawArc

### 描述

在当前视窗的指定坐标按指定尺寸绘一段圆弧，一段圆弧就是一个圆的一部分轮廓。

### 函数原型

```
void GL_DrawArc (int xCenter, int yCenter, int rx, int ry, int a0, int a1);
```

| 参　数 | 含　意 |
|---|---|
| xCenter | 客户视窗中圆弧中心的水平方向坐标（以像素为单位） |
| yCenter | 客户视窗中圆弧中心的垂直方向坐标（以像素为单位） |
| rx | X轴半径（像素）。 |
| ry | Y轴半径（像素）。 |
| a0 | 起始角度（度） |
| a1 | 终止角度（度） |

**限制**

现在不使用参数ry，取而代之的是参数rx。

**附加信息**

GUI_DrawArc使用浮点库。处理的参数rx/ry不能超过180，因为那样将使用到导致溢出错误的整数运算。

**范例**

```
void DrawArcScale(void)
{
    int x0 = 160;
    int y0 = 180;
    int i;
    char ac[4];
    GUI_SetBkColor(GUI_WHITE);
    GUI_Clear();
    GUI_SetPenSize( 5 );
    GUI_SetTextMode(GUI_TM_TRANS);
    GUI_SetFont(&GUI_FontComic18B_ASCII);
    GUI_SetColor( GUI_BLACK );
    GUI_DrawArc( x0,y0,150, 150,-30, 210 );
    GUI_Delay(1000);
    for (i=0; i<= 23; i++)
    {
        float a = (-30+i*10)*3.1415926/180;
        int x = -141*cos(a)+x0;
        int y = -141*sin(a)+y0;
        if (i%2 == 0)
            GUI_SetPenSize( 5 );
        else
            GUI_SetPenSize( 4 );
        GUI_DrawPoint(x,y);
        if (i%2 == 0)
        {
```

```
            x = -123*cos(a)+x0;
            y = -130*sin(a)+y0;
            sprintf(ac, "%d", 10*i);
            GUI_SetTextAlign(GUI_TA_VCENTER);
            GUI_DispStringHCenterAt(ac,x,y);
        }
    }
}
```

**上面范例程序执行结果的屏幕截图**