

第10章 存储设备

存储设备可以用在多种情况下，主要防止显示屏在有对象重叠的绘图操作时的闪烁现象。基本的思路很简单。没有使用存储设备时，绘图操作直接写屏。屏幕在绘图操作在执行时更新，当不同的更新在执行时会产生闪烁。例如，如果你想绘一幅位图作为背景，以一些透明的文字作为前景，你首先必须绘位图，然后是文字，最终结果文字会是闪烁的。


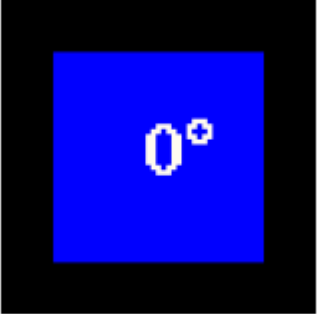
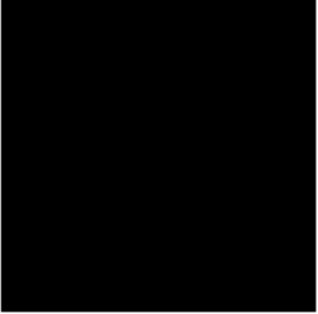
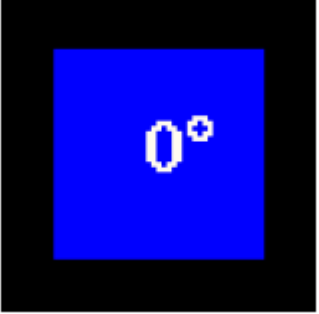
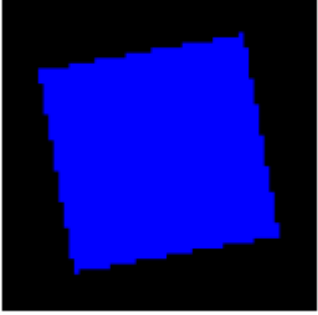
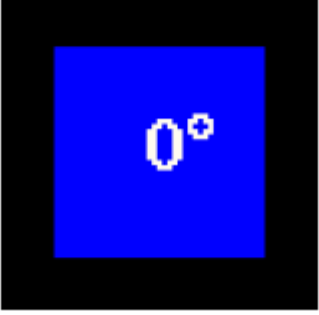
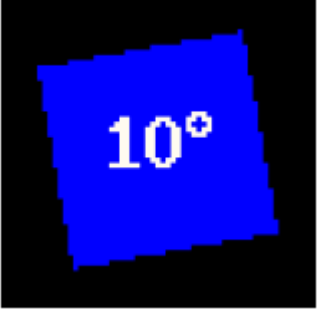
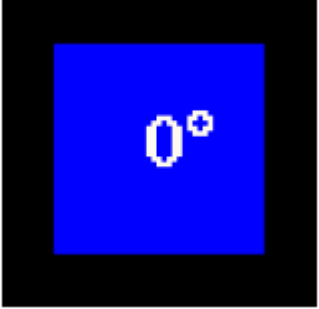
然而，如果这样的操作使用一个存储设备的话，所有的操作在存储设备内执行。只有在所有的操作执行完毕后最终结果才显示在屏幕上，具有无闪烁的优点。二者的不同会在下面的范例中列出，该范例图解一系列绘图操作在没有使用存储设备和使用存储设备的效果。

二者的区别总结如下：如果不使用存储设备，绘图的操作的效果看起来是一步一步的，带来闪烁的缺点。而使用存储设备，所有程序执行的效果看起来象单步操作，没有中间步骤可见。优势在于，如上说明，显示屏的闪烁完全消除，而这常常是希望看到的。

存储设备是一个附加（可选）的软件项目，不随 μ C/GUI 的基本软件包一起发布。存储设备的软件包位于子目录 GUI\Memdev 下。

10.1 图解存储设备的使用

下表是屏幕截图显示使用存储设备和不使用存储设备完成同样操作。两个例子的目的是相同的：旋转一个工件，标注各自的旋转角度（在此处是 10 度）。在第一个例子（不使用存储设备），必须清屏，然后在新的位置重绘矩形和以新的标号写字符串。在第二个例子（使用一个存储设备），同样的操作在存储器执行，但是屏幕在这个时候并没有更新。唯一的更新出现在调用 GUI_MEMDEV_CopyToLCD 函数时，这样的更新立即反映了所有操作。注意两种处理方法的状态初始化和最后的输出是不同的。

API 函数	不使用存储设备	使用存储设备
步骤 1: 状态初始化		
步骤 2: GUI_Clear		
步骤 3: GUI_DrawPolygon		
步骤 4: GUI_DispString		

<p>步骤 5: GUI_MEMDEV_CopyToLCD (只有在使用存储设备时有 效)</p>		
---	--	---

10.2 基本函数

下面的函数是那些在使用存储设备时通常被调用的。基本用法相当简单：

1. 建立存储设备（使用 GUI_MEMDEV_Create）；
2. 激活它（使用 GUI_MEMDEV_Select）；
3. 执行绘图操作；
4. 将结果拷贝到显示屏（使用 GUI_MEMDEV_CopyToLCD）；
5. 如果你不再需要存储设备，删除它（使用 GUI_MEMDEV_Delete）。

10.3 为了能使用存储设备……

默认情形下，存储设备是被激活的。为了优化软件的性能，对存储设备的支持可以在配置文件 GUIConf.h 中加入下面一行而关闭：

```
#define GUI_SUPPORT_MEMDEV    0
```

如果这一行出现在配置文件里，而你又需要使用存储设备，可以删除这一行或将其定义改为 1。

10.4 存储设备 API 函数

下表列出了 μ C/GUI 的存储器 API 函数，所有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
基本函数	
GUI_MEMDEV_Create ()	建立存储设备（第一步）
GUI_MEMDEV_CopyToLCD ()	将存储设备的内容拷贝到 LCD
GUI_MEMDEV_Delete ()	存储设备释放使用的存储空间

GUI_MEMDEV_Select ()	选择一个存储设备作为目标用于绘图操作
高级特性	
GUI_MEMDEV_Clear ()	将存储设备的内容标志为未改变 M
GUI_MEMDEV_CopyFromLCD ()	将 LCD 的内容拷贝到存储设备
GUI_MEMDEV_CopyToLCDAA ()	以反锯齿方式拷贝存储设备的内容
GUI_MEMDEV_GetYSize ()	返回存储设备的 Y 轴尺寸
GUI_MEMDEV_ReduceYSize ()	减少存储设备的 Y 轴尺寸
GUI_MEMDEV_SetOrg ()	在 LCD 上改变存储设备的原点
分片存储设备	
GUI_MEMDEV_Draw ()	使用一个存储设备进行绘图
自动设备对象函数	
GUI_MEMDEV_CreateAuto ()	建立一个自动设备对象。
GUI_MEMDEV_DeleteAuto ()	删除一个自动设备对象。
GUI_MEMDEV_DrawAuto ()	使用一个 GUI_AUTODEV 对象进行绘图

GUI_MEMDEV_Create()

描述

建立一个存储设备

函数原型

```
GUI_MEMDEV_Handle GUI_MEMDEV_Create(int x0, int y0, int XSize, int YSize)
```

参 数	含 意
x0	存储设备的 X 轴坐标
y0	存储设备的 Y 轴坐标
XSize	存储设备的 X 轴尺寸
YSize	存储设备的 Y 轴尺寸

返回数值

建立的存储设备的句柄，如果函数执行失败，返回值为 0。

GUI_MEMDEV_CopyToLCD()

描述

将一个存储设备的内容从内存拷贝到 LCD。

函数原型

```
void GUI_MEMDEV_CopyToLCD(GUI_MEMDEV_Handle hMem)
```

参 数	含 意
hMem	存储设备的句柄

GUI_MEMDEV_Delete()**描述**

删除一个存储设备。

函数原型

```
void GUI_MEMDEV_Delete(GUI_MEMDEV_Handle hMem);
```

参 数	含 意
hMem	存储设备的句柄

返回值

删除的存储设备的句柄

GUI_MEMDEV_Select()**描述**

激活一个存储设备(或如果句柄为 0 则激活 LCD)

函数原型

```
void GUI_MEMDEV_Select(GUI_MEMDEV_Handle hMem)
```

参 数	含 意
hMem	存储设备的句柄

10.5 高级特性

GUI_MEMDEV_Clear()

描述

将存储设备的所有内容标志为“未改变的”。

函数原型

```
void GUI_MEMDEV_Clear(GUI_MEMDEV_Handle hMem);
```

参 数	含 意
hMem	存储设备的句柄

附加信息

使用 GUI_MEMDEV_CopyToLCD 的下一步绘图操作是，只有在 GUI_MEMDEV_Clear 和 GUI_MEMDEV_CopyToLCD 之间有字节改变的情况才进行写操作。

GUI_MEMDEV_CopyFromLCD()

描述

从 LCD 数据（视频存储器）拷贝一个存储设备的内容到存储设备。换句话说，回读 LCD 的内容到存储设备。

函数原型

```
void GUI_MEMDEV_CopyFromLCD(GUI_MEMDEV_Handle hMem);
```

参 数	含 意
hMem	存储设备的句柄

GUI_MEMDEV_CopyToLCDAA()

描述

从存储区域拷贝存储设备的内容（反锯齿）到 LCD。

函数原型

```
void GUI_MEMDEV_CopyToLCDAA(GUI_MEMDEV_Handle hMem);
```

参 数	含 意
<code>hMem</code>	存储设备的句柄

附加信息

器件的数据处理为反锯齿数据。一个 2×2 像素矩阵转换 1 个像素。最终结果的像素强度取决于在多少像素在矩阵中调整。

范例

建立一个存储设备并选择它作为输出。设置一个大的字体，然后向存储设备写入一个文本：

```
GUI_MEMDEV_Handle hMem = GUI_MEMDEV_Create(0, 0, 60, 32);
GUI_MEMDEV_Select(hMem);
GUI_SetFont(&GUI_Font32B_ASCII);
GUI_DispString("Text");
GUI_MEMDEV_CopyToLCDAA(hMem);
```

范例执行的屏幕截图



GUI_MEMDEV_GetYSize()

描述

返回一个存储设备的 Y 轴尺寸。

函数原型

```
int GUI_MEMDEV_GetYSize(GUI_MEMDEV_Handle hMem);
```

参 数	含 意
<code>hMem</code>	存储设备的句柄

GUI_MEMDEV_ReduceYSize()

描述

减小一个存储设备的 Y 轴尺寸。

函数原型

```
void GUI_MEMDEV_ReduceYSize(GUI_MEMDEV_Handle hMem, int YSize);
```

参 数	含 意
<code>hMem</code>	存储设备的句柄
<code>YSize</code>	存储设备新的 Y 轴尺寸

附加信息

改变存储设备的尺寸比删除然后重建它更有效。

GUI_MEMDEV_SetOrg

描述

改变存储设备在 LCD 上的原点。

函数原型

```
void GUI_MEMDEV_SetOrg(GUI_MEMDEV_Handle hMem, int x0, int y0);
```

参 数	含 意
<code>hMem</code>	存储设备的句柄
<code>x0</code>	水平坐标（左上角像素）
<code>y0</code>	垂直坐标（左上角像素）

附加信息

这个函数在同一个器件用于不同的屏幕区域或存储设备的内容被拷贝到不同区域时非常有用。

修改存储设备的原点比删除然后重建它更有效。

使用存储设备的简单例子

该范例展示一个基本存储设备的使用。向存储设备写入一个文本，然后再拷贝到 LCD。该范例见 Source\Misc\MemDev.c 文件。

```

/*-----
文件:      MemDev.c
目的:      展示如何使用存储设备的简单例子
-----*/

#include "GUI.H"

/*****
*                      展示存储设备的使用                      *
*****/

static void DemoMemDev(void)
{
    GUI_MEMDEV_Handle hMem;
    while(1)
    {
        /* 建立存储设备..... */
        hMem = GUI_MEMDEV_Create(0, 0, 110, 18);
        /* .....然后选择其用于绘图操作 */
        GUI_MEMDEV_Select(hMem);
        /* 向存储设备绘一个文本*/
        GUI_SetFont(&GUI_FontComic18B_ASCII);
        GUI_DispStringAt("Memory device", 0, 0);
        /* 将存储设备的内容拷贝到 LCD */
        GUI_MEMDEV_CopyToLCD(hMem);
        /* 选择 LCD 并清除存储设备 */
        GUI_MEMDEV_Select(0);
        GUI_MEMDEV_Delete(hMem);
        GUI_Delay(1000);
        GUI_Clear();
        GUI_Delay(500);
    }
}

```

```

/*****
*                               *
*                               *
*****/

```

```

void main(void)
{
    GUI_Init();
    DemoMemDev();
}

```

10.6 分片存储设备

一个存储设备首先通过执行指定的绘图函数进行内容填充。设备填充完毕后，其内容写入 LCD。有些情况下，可能会没有足够的有效存储器空间能够立刻用于所有输出区域的存储，这依赖于你的配置（参考第 21 章“高层次配置”的 GUI_ALLOC_SIZE 配置宏）。一个分片存储设备将绘制区域分成几个片段，在每一片段里面用尽可能多占用当前可用的存储空间。

GUI_MEMDEV_Draw()

描述

防止显示屏闪烁的基本函数

函数原型

```

int GUI_MEMDEV_Draw ( GUI_RECT* pRect,
                      GUI_CALLBACK_VOID_P* pfDraw,
                      void* pData,
                      int NumLines,
                      int Flags)

```

参 数	含 意
<code>pRect</code>	所使用的 LCD 的一个 GUI_RECT 结构的指针。
<code>pfDraw</code>	执行绘图操作的一个回调函数的指针。
<code>pData</code>	作为回调函数参数使用的一个数据结构的指针。
<code>NumLines</code>	0（推荐）或者是存储设备的分段数量。
<code>Flags</code>	0 或者 GUI_MEMDEV_HASTRANS。

返回数值

如果成功返回 0，如果函数执行失败则返回 1。

附加信息

如果参数 NumLines 为 0，则每段中的线段数量由函数自动计算。通过移动存储设备的原点，函数一段又一段地重复改写输出区域。

使用分片存储设备的范例

下面的范例展示了一个分片存储设备的使用。其源文件是 Source\Misc\BandingMemdev.c。

```

/*-----
文件:      BandingMemdev.c
目的:      展示如何使用分片存储设备的例子
-----*/

#include "GUI.H"
static const GUI_POINT aPoints[] =
{
    {-50, 0}, {-10, 10}, {0, 50}, {10, 10}, {50, 0}, {10, -10}, {0, -50}, {-10, -10}
};
#define SIZE_OF_ARRAY (Array) (sizeof(Array)/sizeof(Array[0]))
typedef struct
{
    int XPos_Poly, YPos_Poly; int XPos_Text, YPos_Text; GUI_POINT aPointsDest[8];
} tDrawItContext;

/******
*                      绘图函数                      *
*****/

static void DrawIt(void * pData)
{
    tDrawItContext * pDrawItContext = (tDrawItContext *)pData;
    GUI_Clear();
    GUI_SetFont(&GUI_Font8x8);

```

```

GUI_SetTextMode(GUI_TM_TRANS);
/* 绘背景 */
GUI_SetColor(GUI_GREEN);
GUI_FillRect ( pDrawItContext->XPos_Text,
               pDrawItContext->YPos_Text - 25,
               pDrawItContext->XPos_Text + 100,
               pDrawItContext->YPos_Text - 5);
/* 绘多边形 */
GUI_SetColor(GUI_BLUE);
GUI_FillPolygon (pDrawItContext->aPointsDest,
                 SIZE_OF_ARRAY(aPoints),
                 160, 120);
/* 绘前景 */
GUI_SetColor(GUI_RED);
GUI_FillRect( 220 - pDrawItContext->XPos_Text,
              pDrawItContext->YPos_Text + 5,
              220 - pDrawItContext->XPos_Text + 100,
              pDrawItContext->YPos_Text + 25);
}

/*****
*                               展示分片存储设备                               *
*****/
#define USE_BANDING_MEMDEV (1)
/* 如设为 0，则定义不使用分片存储设备进行绘图 */
void DemoBandingMemdev(void)
{
    int i;
    int XSize = LCD_GET_XSIZE();
    int YSize = LCD_GET_YSIZE();
    tDrawItContext DrawItContext;
    GUI_SetFont (&GUI_Font8x9);
    GUI_SetColor(GUI_WHITE);
    GUI_DispStringHCenterAt ( "Banding memory device\nwithout flickering",
                             XSize/40)

    DrawItContext.XPos_Poly = Xsize/2;
    DrawItContext.YPos_Poly = Ysize/2;

```

```

DrawItContext.YPos_Text = Ysize/2-4;
for(i = 0; i < (XSize - 100); i++)
    float angle = i * 3.1415926 / 60;
DrawItContext.XPos_Text = i;
/* 旋转多边形 */
GUI_RotatePolygon ( DrawItContext.aPointsDest,
                    aPoints,
                    SIZE_OF_ARRAY(aPoints),
                    angle);
#if USE_BANDING_MEMDEV
{
    GUI_RECT Rect = { 0, 70, 320, 170} ;
    /* 使用分片存储设备进行绘图 */
    GUI_MEMDEV_Draw(&Rect, &DrawIt, &DrawItContext, 0, 0);
}
#else
/* 不使用存储设备的简单绘图 */
    DrawIt((void *)&DrawItContext);
#endif
#ifdef WIN32
    GUI_Delay(20); /* 只有在仿真时才使用一段短延时 */
#endif
}

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

void main (void)
{
    GUI_Init () ;
    while(1)
    {
        DemoBandingMemdev ();
    }
}

```

范例执行结果的屏幕截图



10.7 自动设备对象

当显示屏必须更新以反映其对象的移动或改变时，存储设备非常有用，因此在防止 LCD 闪烁这样一个应用方面是很重要的。一个自动设备对象是基于分片存储设备建立的，它可以在某些应用方面更有效，例如移动标志，这种情况下，在一段时间内显示屏只有一小部分要更新。

该设备能自动识别显示屏的哪一部分包含固定的对象，哪一部分包含移动或改变的对象（必须更新）。当绘图函数第一次被调用时，所有的对象都被绘制出。而以后的函数调用只更新需要移动或改变的物体。实际的绘图操作使用分片存储设备机制，但只在需要的的空间内使用。使用一个自动存储设备（与直接使用分片存储设备相比）的主要优点是节省了计算时间，因为它需要更新整个显示屏。

GUI_MEMDEV_CreateAuto()

描述

建立一个自动设备对象。

函数原型

```
int GUI_MEMDEV_CreateAuto(GUI_AUTODEV * pAutoDev);
```

参 数	含 意
<code>pAutoDev</code>	一个 GUI_AUTODEV 对象的指针

返回数值

通常是 0，保留供以后使用。

GUI_MEMDEV_DeleteAuto()**描述**

删除一个自动设备对象。

函数原型

```
void GUI_MEMDEV_DeleteAuto(GUI_AUTODEV * pAutoDev);
```

参 数	含 意
<code>pAutoDev</code>	一个 <code>GUI_AUTODEV</code> 对象的指针

GUI_MEMDEV_DrawAuto()**描述**

使用一个分片存储设备执行一个指定的绘图函数。

函数原型

```
int GUI_MEMDEV_DrawAuto ( GUI_AUTODEV * pAutoDev,
                          GUI_AUTODEV_INFO * pAutoDevInfo,
                          GUI_CALLBACK_VOID_P * pfDraw, void * pData);
```

参 数	含 意
<code>pAutoDev</code>	一个 <code>GUI_AUTODEV</code> 对象的指针。
<code>pAutoDevInfo</code>	一个 <code>GUI_AUTODEV_INFO</code> 对象的指针。
<code>pfDraw</code>	用户定义要执行的绘图函数的指针。
<code>pData</code>	一个由绘图函数传递的数据结构的指针。

返回数值

如果成功返回 0，函数执行失败返回 1

附加信息

GUI_AUTODEV_INFO 结构包含有哪些对象必须要由用户函数绘制的信息:

```
typedef struct
{
    char DrawFixed;
} GUI_AUTODEV_INFO;
```

如果所有的对象都要绘制, DrawFixed 设为 1。当只有被移动或改变的物体才需要绘制的时候, 设为 0。当使用这个特性时, 我们推荐使用下面的程序:

```
typedef struct
{
    GUI_AUTODEV_INFO AutoDevInfo;      /* 哪些内容需要绘制的信息 */
    /* 给用户函数添加使用的数据 */
    .....
} PARAM;

static void Draw(void * p)
{
    PARAM * pParam = (PARAM *)p;
    if (pParam->AutoDevInfo.DrawFixed)
    {
        /* 绘固定的背景 */
        .....
    }
    /* 绘制移动的物体 */
    .....
    if (pParam->AutoDevInfo.DrawFixed)
    {
        /* 绘固定的前景 (如果需要) */
        .....
    }
}

void main (void)
{
```



```

PARAM Param;                /* 绘图函数的参数*/
GUI_AUTODEV AutoDev;        /* 分片存储设备的对象 */
/* 绘图函数的 设置/修改 信息 */
.....
GUI_MEMDEV_CreateAuto(&AutoDev); /* 建立 GUI_AUTODEV 对象 */
GUI_MEMDEV_DrawAuto ( &AutoDev, /* 使用 GUI_AUTODEV 对象用于绘图 */
                      &Param.AutoDevInfo,
                      &Draw, &Param);
GUI_MEMDEV_DeleteAuto(&AutoDev); /* 删除 GUI_AUTODEV 对象 */
}

```

使用一个自动设备对象的范例

下面的范例展示了一自动设备对象的使用。其源文件为：Source\Misc\AutoDev.c。在背景上绘一个带有可转动指针的刻度盘，在前景上绘一段小的文字。指针使用 μ C/GUI 的抗锯齿特性绘制。在这里使用高分辨率抗锯齿以增强转动的指针的外观效果。对于抗锯齿的更多信息，请参阅第 15 章：抗锯齿。

```

/*-----
文件:      AutoDev.c
目的:      展示 GUI_AUTODEV 对象用法的例子
-----*/

#include "GUI.H"
#include <math.h>
#include <stddef.h>
#ifndef WIN32
    #include "rtos.h"
#endif
#define countof (Obj) (sizeof(Obj)/sizeof (Obj[0]))
#define DEG2RAD (3.1415926f/180)

/*****
*                               *
*                               *
*****/

static const GUI_COLOR ColorsScaleR140[] =
{
    0x000000, 0x00AA00, 0xFFFFFf, 0x0000AA, 0x00FF00, 0xAEAEAE, 0x737373,

```

```

    0xD3D3D3, 0xDFDFDF, 0xBBDFBB, 0x6161DF, 0x61DF61, 0BBBBDF, 0xC7C7C7,
    0x616193
};
static const GUI_LOGPALETTE PalScaleR140 =
{
    15,                /* number of entries */
    0,                 /* 不透明 */
    &ColorsScaleR140[0]
};
static const unsigned char acScaleR140[] =
{
    /* ..... 像素数据不在菜单中显示。 请参考范例源文件..... */
};
static const GUI_BITMAP bmScaleR140 =
{
    200,                /* X 轴尺寸 */
    73,                 /* Y 轴尺寸 */
    100,                /* 每行字节数 */
    4,                  /* 每像素位数 */
    acScaleR140,        /* 图片数据的指针 (像素) */
    &PalScaleR140       /* 调色板的指针 */
};

/*****
*                               *
*                               *
*****/

#define MAG 3
static const GUI_POINT aNeedle[] =
{
    {MAG * (0) , MAG * (0 + 125)},
    {MAG * (-3), MAG * (-15 + 125)},
    {MAG * (-3), MAG * (-65 + 125)},
    {MAG * (3), MAG * (-65 + 125)},
    {MAG * (3), MAG * (-15 + 125)},
};

```

```

/*****
*
*                               包括绘图函数信息的结构
*
*****/

typedef struct
{
    /* Information about what has to be displayed */
    GUI_AUTODEV_INFO AutoDevInfo;
    /* 多边形数据 */
    GUI_POINT aPoints[7] ;
    float Angle;
} PARAM;

/*****
*
*                               获得角度
*
*****/

/* 这个函数返回数值进行显示。在实际应用中，该数值以某种方式进行测量 */
static float GetAngle(int tDiff)
{
    if (tDiff < 15000)
    {
        return 225 - 0.006 * tDiff ;
    }
    tDiff -= 15000;
    if (tDiff < 7500)
    {
        return 225 - 90 + 0.012 * tDiff ;
    }
    tDiff -= 7000;
    return 225;
}

/*****
*
*                               绘图函数
*
*****/

```

```

static void Draw(void * p)
{
    PARAM * pParam = (PARAM *)p;
    /* 固定前景 Fixed background */
    if (pParam->AutoDevInfo.DrawFixed)
    {
        GUI_ClearRect ( 60,
                        50 + bmScaleR140.YSize,
                        60 + bmScaleR140.XSize - 1,
                        150);
        GUI_DrawBitmap(&bmScaleR140, 60, 50);
    }
    /* 绘指针 */
    GUI_SetColor(GUI_WHITE);
    GUI_AA_FillPolygon(pParam->aPoints, countof(aNeedle), MAG*160, MAG*190);
    /* 固定前景 Fixed foreground */
    if (pParam->AutoDevInfo.DrawFixed)
    {
        GUI_SetTextMode(GUI_TM_TRANS);
        GUI_SetColor(GUI_RED);
        GUI_SetFont(&GUI_Font24B_ASCII);
        GUI_DispStringHCenterAt("RPM/1000", 160, 110);
    }
}

/*****
*                               使用分片存储设备显示一个带指针的刻度盘                               *
*****/

static void DemoScale(void)
{
    int Cnt;
    int tDiff, t0 = GUI_GetTime();
    PARAM Param;                               /* 绘图函数的参数 */
    GUI_AUTODEV AutoDev;                       /* 分片存储设备对象 */
    /* 显示消息 */
    GUI_SetColor(GUI_WHITE);

```

```

GUI_SetFont (&GUI_Font8x16);
GUI_DispStringHCenterAt ("Scale using GUI_AUTODEV-object", 160, 0);
/* 启动高分辨率用于抗锯齿 */
GUI_AA_EnableHiRes ();
GUI_AA_SetFactor (MAG);
/* 建立 GUI_AUTODEV 对象 */
GUI_MEMDEV_CreateAuto (&AutoDev);
/* 显示在一个固定时间上的指针 */
for (Cnt = 0; (tDiff = GUI_GetTime () - t0) < 24000; Cnt++)
{
/* 获得数值用于显示一个多边形来表示指针 */
Param.Angle = GetAngle (tDiff)* DEG2RAD;
GUI_RotatePolygon ( Param.aPoints,
                    aNeedle,
                    countof (aNeedle),
                    Param.Angle);
GUI_MEMDEV_DrawAuto (&AutoDev, &Param.AutoDevInfo, &Draw, &Param);
}
/* 显示 "milliseconds / picture" */
GUI_SetColor (GUI_WHITE);
GUI_SetFont (&GUI_Font8x16);
GUI_DispStringHCenterAt ("Milliseconds / picture:", 160, 180);
GUI_SetTextAlign (GUI_TA_CENTER);
GUI_SetTextMode (GUI_TM_NORMAL);
GUI_DispNextLine ();
GUI_GotoX (160);
GUI_DispFloatMin ((float) tDiff / (float) Cnt, 2);
/* 删除 GUI_AUTODEV 对象 */
GUI_MEMDEV_DeleteAuto (&AutoDev);
}

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

void main (void)
{

```

```
#ifndef WIN32
    OS_InitKern();
    OS_InitHW();
#endif
GUI_Init();
while(1)
    DemoScale();
}
```

范例程序执行结果的屏幕截图

