mGEff 编程指南

目录

修订记录

版本

作者	完成日期	评审者	评审日期	版本
董凯 王旭斌 胡兆麟	2011.02.14	-	-	V1.0

Edit

变更记录

版本	日期	内容	备注
V1.0	2011.02.14	完成初稿	mGEff V1.0 - rev 1642

-- XuBinWang - 12 Jan 2011

This topic: Main > MiniGUI > MGEffV1dot0ProgrammingGuide
History: r23 - 15 Feb 2011 - 19:10:36 - YongmingWei

版权声明

版权声明

《mGEff 编程指南》版本 1.0，适用于 mGEff Version 1.0.x。

-- ZhaolinHu - 18 Jan 2011

This topic: Main > MiniGUI > MGEffV1dot0ProgrammingGuide > MGEffV1dot0PGC00CopyrightClaim

History: r3 - 11 Feb 2011 - 11:09:16 - XuBinWang

关于本编程指南

章节编排

- 第一部分 mGEff 简介 ------- 介绍了动画的原理以及mGEff的框架等。
- 第二部分 mGEff 编程基础 ------- 本部分将从最简单的例子开始，完整的向读者介绍 mGEff 动画的基础接口与生命周期等。
- 第三部分 mGEff 高级动画 ------- 本部分介绍 mGEff 内建特效器的使用和主窗口动画，mGEff 提供了大量稳定而高效的特效器，为读者快速实现翻转，放大，滚屏，卷页等常用动画提供了可能。并且mGEff可以与MiniGUI结合，以双缓冲为基础开发了针对主窗口动画的动画接口。
- 第四部分 mGEff 编程进阶 --- 本部分属于高阶编程，介绍如何自定义动力曲线与特效器，并通过重载各种默认的动画回调以更有力的控制动画执行，协助读者随心所欲的实现高效而富有个性的动画

-- KaiDong - 19 Jan 2011

This topic: Main > MiniGUI > MGEffV1dot0ProgrammingGuide > MGEffV1dot0PGC00AboutMe
History: r4 - 27 Jan 2011 - 01:29:40 - XuBinWang

mGEff 前言

前言

-- XuBinWang - 12 Jan 2011

This topic: Main > MiniGUI > MGEffV1dot0ProgrammingGuide > MGEffV1dot0PGC00
History: r4 - 11 Feb 2011 - 11:08:57 - XuBinWang

动画原理

动画的原理

动画是将静止的画面变为动态的艺术，实现由静止到动态的变化，主要依靠的是人眼的视觉残留效应，一般而言，动画分为两种类型：帧 (Frame) 动画和属性 (Property) 动画。

帧动画

帧动画是指播放事先准备好的一系列存在一定差异的图像来实现图像上事物动态变化的一种动画，比如GIF图片等；

属性动画

属性动画则是指通过改变被显示对象的各种属性（例如尺寸、位置、角度、透明度等）来达到变化效果的一种动画，这种动画也有帧的概念，只不过每一帧是通过实时计算对象的属性值来产生的。

mGEff支持的是属性动画

mGEff 支持的是属性动画：

用户通过设置动画对象相关属性的一些参数，比如属性的起始值、终止值、动力曲线(变化方式)以及动画持续的时间等相关参数，然后mGEff就可以根据这些相关的参数，随时间的变化计算出一组从起始值到终止值变化的数值，按这组时间属性值的变化曲线设置动画对象的相关属性，即可产生相应的动画效果。

归纳起来，mGEff实现了一种在一定时间范围内，根据给定的动力曲线将属性值从起始值变化到终止值的动画机制。在下一章。我们将详细介绍mGEff是如何实现属性动画的。

-- XuBinWang - 12 Jan 2011

This topic: Main > MiniGUI > MGEffV1dot0ProgrammingGuide > MGEffV1dot0PGC01
History: r15 - 27 Jan 2011 - 01:29:41 - XuBinWang

mGEff的架构与特性

mGEff 是什么

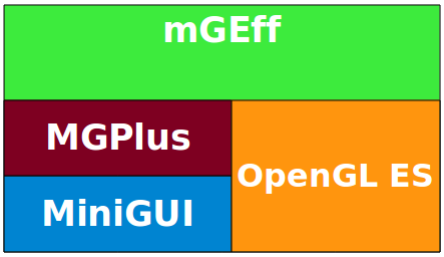
mGEff 是一个轻量级动画框架，它提供了一套高效灵活的动画实现方案。

设计目标

mGEff的设计目标是建立起一套平台无关的动画机制，它可以和MiniGUI、GTK+等GUI系统相结合开发相应的（GDI 级、控件级和窗口级）动画。

mGEff 的框架

以下是它与 MiniGUI 相结合的框架：



mGEff 的类

mGEff 本身采用 C 语言开发的，但其设计及实现上运用了一些基本的面向对象思想，比如以 struct 模拟 class等，了解这些结构（类）对使用它会很有帮助，下面是 mGEff 的“类“图：

// TODO 类图

主要特性

- mGEff 提供了以下主要特性：
  - 驱动动画
  - 动力曲线
  - 串行、并行动画组
  - 特效器
  - 主窗体动画

本章小结

本章简单介绍了 mGEff 的框架结构和主要特性。接下来的章节将会就这些特性逐步展开进行详细的介绍。

-- XuBinWang - 12 Jan 2011

I	Attachment	Action	Size	Date	Who	Comment
	mGEff.png	manage	10.1 K	19 Jan 2011 - 15:32	KaiDong	

r20 - 14 Feb 2011 - 11:12:58 - XuBinWang

开始启程

使用mGEff库

在上一章中我们简单介绍了下mGEff的框架和特性，在接下来的章节，将带领大家进入奇妙的动画世界，首先我们先介绍下mGEff库的使用。

函数原型

mGEff库提供了mGEffInit/mGEffDeinit函数分别负责库的初始化和注销。其原型为：

初始化mGEff库函数

```
/* 初始化mGEff库函数 */
int mGEffInit (void);
```

- 参数说明：
  - void - 无参数；
- 返回值说明：
  - int - 0 初始化成功；

注销mGEff库函数

```
/* 注销mGEff库函数 */
void mGEffDeinit (void);
```

- 参数说明：
  - void - 无参数；
- 返回值说明：
  - void - 无返回值；

使用示例

下面我们结合一个简单的示例来给大家讲解下mGEff库的使用。在这里我们将只关注

mGEffInit/mGEffDeinit这两个函数的使用，其他函数的使用将在后续章节中讲解：

```
#include <stdio.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/minigui.h>

#include <mgeff/mgeff.h>

/*********************************************************************
#define CAPTION    "animation_sync"
#define BAR_HEIGHT 50
#define DURATION   (20 * 1000)
#define START_VAL  0x00
#define END_VAL    0xFF

/*********************************************************************
static char g_str[64];
static int g_value = 0x00;

/*********************************************************************
/* main window proc */
static int mainWindowProc (HWND hWnd, int message, WPARAM wParam, LPAR
/* draw a frame */
static void draw_frame (HWND hWnd);
/* callback function called when property change */
static void property_callback (MGEFF_ANIMATION handle, HWND hWnd, int
/* create and run an animation */
static int do_animation (HWND hWnd);

/*********************************************************************
int MiniGUIMain (int argc, const char *argv[])
{
        HWND hMainHwnd;
        MAINWINCREATE createInfo;
        MSG msg;

        #ifdef _MGRM_PROCESSES
        JoinLayer (NAME_DEF_LAYER, "animation", 0, 0);
        #endif

        createInfo.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
        createInfo.dwExStyle = WS_EX_NONE;
        createInfo.spCaption = CAPTION;
        createInfo.hMenu = 0;
```

```
        createInfo.hCursor = GetSystemCursor (0);
        createInfo.hIcon = 0;
        createInfo.MainWindowProc = mainWindowProc;
        createInfo.lx = 0;
        createInfo.ty = 0;
        createInfo.rx = 240;
        createInfo.by = 320;
        createInfo.iBkColor = COLOR_lightwhite;
        createInfo.dwAddData = 0;
        createInfo.hHosting = HWND_DESKTOP;

        hMainHwnd = CreateMainWindow (&createInfo);

        if (hMainHwnd == HWND_INVALID) {
                return -1;
        }

        ShowWindow (hMainHwnd, SW_SHOWNORMAL);

        while (GetMessage (&msg, hMainHwnd)) {
                TranslateMessage (&msg);
                DispatchMessage (&msg);
        }

        MainWindowThreadCleanup (hMainHwnd);

        return 0;
}

/************************************************************************
/* main window proc */
static int mainWindowProc (HWND hWnd, int message, WPARAM wParam, LPAR
{
        switch (message) {
                case MSG_CREATE:
                /* init mGEff library */
                mGEffInit ();

                sprintf (g_str, "you can press any key.");

                do_animation (hWnd);
                break;

                case MSG_KEYDOWN:
                sprintf (g_str, "you press key, keycode(%d)", wParam);

                printf ("%s\n", g_str);
```

```
                InvalidateRect (hWnd, NULL, TRUE);
                break;

                case MSG_PAINT:
                draw_frame (hWnd);
                break;

                case MSG_CLOSE:
                DestroyMainWindow (hWnd);
                PostQuitMessage (hWnd);

                /* deinit mGEff library */
                mGEffDeinit ();
                break;

                default:
                break;
        }

        return DefaultMainWinProc (hWnd, message, wParam, lParam);
}

/* draw a frame */
static void draw_frame (HWND hWnd)
{
        HDC dc;
        RECT rc;

        int client_w, client_h;
        char str[64];
        int color;

        /* begin draw */
        dc = BeginPaint (hWnd);

        /* get client rect */
        GetClientRect (hWnd, &rc);

        client_w = RECTW (rc);
        client_h = (RECTH (rc) - BAR_HEIGHT) * g_value / (END_VAL - ST

        color = g_value;

        /* set brush and draw area */
        SetBrushColor (dc, RGB2Pixel (dc, color, 0x00, 0x00));

        FillBox (dc, 0, BAR_HEIGHT, client_w, client_h);
```

```
        /* draw the text */
        sprintf (str, "current color: RGB(%d,0,0)", color);

        TextOut (dc, 0, 0, str);

        /* print information */
        TextOut (dc, 0, BAR_HEIGHT / 2, g_str);

        EndPaint (hWnd, dc);
        /* end draw */
}

/* callback function called when property change */
static void property_callback (MGEFF_ANIMATION handle, HWND hWnd, int
{
        /* set animation property */
        g_value = *value;

        /* update */
        InvalidateRect (hWnd, NULL, TRUE);
}

/* create and run an animation */
static int do_animation (HWND hWnd)
{
        MGEFF_ANIMATION animation;
        int animation_id = 1;

        int duration;
        int start_val;
        int end_val;

        /* set value */
        duration = DURATION;
        start_val = START_VAL;
        end_val = END_VAL;

        /* create animation object */
        animation = mGEffAnimationCreate ((void *) hWnd, (void *) prop

        /* set animation property */
        /* duration */
        mGEffAnimationSetDuration (animation, duration);

        /* start value */
        mGEffAnimationSetStartValue (animation, &start_val);

        /* end value */
```

```
        mGEffAnimationSetEndValue (animation, &end_val);

        /* running */
        mGEffAnimationSyncRun (animation);

        /* delete the animation object */
        mGEffAnimationDelete (animation);

        return 0;
}
```

可以看到，程序在创建的时候（消息MSG_CREATE中）调用了初始化函数mGEffInit()，在关闭的时候（消息MSG_CLOSE中）调用了注销函数mGEffDeinit()。在这两个函数之间调用诸如创建动画，设置属性，运行动画等mGEff函数。

运行结果

注意事项

但需要注意的是，mGEff 不是线程安全的，所以请不要在多个线程中同时使用。

本章小结

本章通过一个简单的 mGEff 示例演示了如何在 MiniGUI 程序中引入和准备 mGEff 环境，做好了这些准备工作，就能够开始使用 mGEff 实现动画了。

-- XuBinWang - 12 Jan 2011

创建和运行动画

创建和运行动画

在上一章中我们介绍了mGEff库的初始化和注销，通过例子我们学习了库的初始化和注销函数在什么地方什么时候调用。在这章里面我们将要学习如何创建一个动画对象以及使这个动画动起来。

函数原型

mGEff库提供了两个函数分别用来创建动画对象和运行动画。其原型为：

创建动画对象函数

BeautifierPlugin Error: Unable to handle "“cpp”" language.

```
/* 创建动画对象 */
MGEFF_ANIMATION mGEffAnimationCreate (void *target, MGEFF_SETPROPERTY_
```

- 参数说明：
  - target - 要进行动画的目标对象，这里因为我们只演示mGEff创建动画的方式，所以先暂时传入 NULL，没有具体的动画对象；
  - setproperty - 属性变化时要调用的回调函数指针，此函数会在动画的每一帧产生时被调用，稍后会着重进行介绍；
  - id - 属性/动画的 ID，这个值将作为参数传递给回调函数，用来在多个动画共用同一回调函数时区分要修改的属性；
  - varianttype - 属性值的类型，这个参数决定了属性起始值与终止值的数据类型，mGEff里定义了一组类型的枚举，其定义如下：

BeautifierPlugin Error: Unable to handle "“cpp”" language.

```
enum EffVariantType {
    MGEFF_INT       = 0,   /* 整型          */
    MGEFF_FLOAT     = 1,   /* 浮点类型      */
    MGEFF_DOUBLE    = 2,   /* 双精度浮点类型 */
    MGEFF_POINT     = 3,   /* 整型点        */
```

```
    MGEFF_POINTF    = 4,   /* 浮点类型点     */
    MGEFF_3DPOINT   = 5,   /* 整型3维点      */
    MGEFF_3DPOINTF  = 6,   /* 浮点类型3维点  */
    MGEFF_RECT      = 7,   /* 区域           */
    MGEFF_COLOR     = 8,   /* 颜色           */
    MGEFF_MAX
};
```

- 返回值说明：
  - handle - 动画对象句柄；

回调函数

mGEff是属性动画，而当动画的属性值（比如当前帧、状态、方向等）改变的时候，mGEff会通过调用回调的方式来通知用户。比如这里的MGEFF_SETPROPERTY_CB回调函数就会在每一帧发生的时候调用，用户可以在这里根据属性的变化值绘制动画对象。　　这里我们先接触一下帧回调函数，关于回调函数的概念我们将有专门的一章来介绍它，回调函数的原型： **BeautifierPlugin Error: Unable to handle "“cpp”" language.**

```
typedef void (* MGEFF_SETPROPERTY_CB)(MGEFF_ANIMATION handle, void *ta
```

- 参数说明：
  - handle - 调用本函数的动画句柄，通过这个句柄可以在函数中访问到与动画相关的一些信息；
  - target - 要进行动画的目标对象，它在调用 mGEffAnimationCreate 创建动画时被指定，它可以是窗口、控件句柄或者是自定义的对象；
  - id - 属性的 ID 值，同样也是在调用 mGEffAnimationCreate 时指定的，有时用户希望在多个动画里共用同一个回调函数以控制对象多个属性的变化，此时可以为每个属性设置不同的 ID 值，让用户在回调函数里判断当前发生改变的属性。针对一些常用的属性对象，预先设置了如下 ID 值，用户可以直接拿来用：

BeautifierPlugin Error: Unable to handle "“cpp”" language.

```
   enum EffPropertyID {
    ID_NULL = 0,
    ID_POS, /* POINT */
    ID_SIZE, /* RECT */
    ID_RECT, /* RECT, pos & size */
    ID_OPACITY, /* int */
    ...
    ID_USER = 1000;
};
```

  - value - 属性的当前值，属性的起始值与终止值由 mGEffAnimationSetStartValue/mGEffAnimationSetEndValue 两个函数设置，在前面提到过 mGEff 的整个动画过程就是将属性从起始值变化到终止值，而这个 value，每一帧的变化值。本示例中只是将该值做打印处理，用户可以用它来驱动动画对象的相关属性，比如长度，大小，颜色等，来实现相关的动画效果。

- 返回值说明：
  - void - 无返回值；

同步运行动画对象函数

BeautifierPlugin Error: Unable to handle "“cpp”" language.

```
/* 同步运行动画 */
int mGEffAnimationSyncRun (MGEFF_ANIMATION handle);
```

- 参数说明：
  - handle - 动画对象句柄
- 返回值说明：
  - int -

使用示例

仍然以上一章中的示例为例，函数do_animation中包含了创建和运行一个动画的全过程：

BeautifierPlugin Error: Unable to handle "“cpp”" language.

```
/* callback function called when property change */
static void property_callback (MGEFF_ANIMATION handle, HWND hWnd, int
{
    /* set animation property */
    g_value = *value;

    /* update */
    InvalidateRect (hWnd, NULL, TRUE);
}

/* create and run an animation */
static int do_animation (HWND hWnd)
{
    MGEFF_ANIMATION animation;
    int animation_id = 1;

    int duration;
    int start_val;
    int end_val;

    /* set value */
    duration = DURATION;
    start_val = START_VAL;
    end_val = END_VAL;

    /* create animation object */
    animation = mGEffAnimationCreate ((void *) hWnd, (void *) property

    /* set animation property */
```

```
    /* duration */
    mGEffAnimationSetDuration (animation, duration);

    /* start value */
    mGEffAnimationSetStartValue (animation, &start_val);

    /* end value */
    mGEffAnimationSetEndValue (animation, &end_val);

    /* running */
    mGEffAnimationSyncRun (animation);

    /* delete the animation object */
    mGEffAnimationDelete (animation);

    return 0;
}
```

通过阅读上述代码，我们看到创建一个动画对象的步骤为：先创建了一个动画对象，设置帧回调函数。然后对这个动画对象，设置了相关属性的起始值、结束值和动画持续时间等。最后调用动画运行函数，运行这个动画。

运行结果

句柄的概念

从上面的代码可以看到，正如 MiniGUI 一样，mGEff 对外提供的对象访问方式也是通过“句柄”来完成的。句柄与普通指针的区别在于：

指针包含的是引用对象的内存地址，而句柄则是由框架所管理的引用标识，该标识可以被框架重新定位到一个内存地址上。这种间接访问对象的模式增强了系统对引用对象的控制，并减少对外暴露的数据结构，有利于保持一个较简洁的 API 接口层。

本章小结

本章展示了一个最简单的动画是如何创建、运行的，并附带了一个简单的示例，在之后的章节里，会逐渐扩展完善这个示例以便向用户演示更多的 mGEff 特性。

另外这一章详细描述了属性变化回调函数的概念和用法，通过实现不同的属性变化回调，用户可以得到各种各样的动画特效，概括起来就是 mGEff 只提供了运行动画的驱动力，具体要如何运用这个驱动力来产生动画，主要就看用户如何实现这个回调函数了。

最后还介绍了“句柄”一词在 mGEff 中的含义，通过句柄访问对象是 mGEff 提供统一访问途径。

-- XuBinWang - 12 Jan 2011

r11 - 14 Feb 2011 - 09:52:01 - XuBinWang

动画的参数

动画的参数

在上一章中我们介绍了如何创建动画以及如何运行这个动画，接下来我们将介绍如何去控制动画的运行时间以及动画对象相关属性值的控制比如起始值和结束值等。

函数原型

mGEff提供了三个函数分别用于控制动画的运行时间和设置动画属性值的起始值和结束值。其原型分别是：

设置动画运行时间函数

```
/* 设置动画运行时间 */
void mGEffAnimationSetDuration (MGEFF_ANIMATION handle, int duration_m
```

- 参数说明：
  - handle - 动画对象句柄；
  - duration_ms - 运行时间，单位ms；
- 返回值说明：
  - void - 无返回值；

设置动画属性开始值函数

```
/* 设置动画开始值 */
void mGEffAnimationSetStartValue (MGEFF_ANIMATION handle, const void *
```

- 参数说明：
  - handle - 动画对象句柄；
  - value - value 是 void* 型，这就允许往里面传入任意类型的指针。这么设计的原因是动画要控制的属性可能有多种类型，例如整型、浮点型甚至是结构体，所以这里并不限定具体的类型。而这一类型将在动画创建时指定，具体请参考《创建和运行动画》一章中对 mGEffAnimationCreate 函数的介绍。在调用上述两个函数设置属性的起止值时，应

该传入创建动画时所约定类型的指针，否则会导致不可预料的程序错误！

- 返回值说明：
  - void - 无返回值；

设置动画属性结束值函数

```
/* 设置动画结束值 */
void mGEffAnimationSetEndValue (MGEFF_ANIMATION handle, const void *va
```

- 参数说明：
  - handle - 动画对象句柄；
  - value - 和传入mGEffAnimationSetStartValue的值的类型一致；
- 返回值说明：
  - void - 无返回值；

使用示例

仍以之前的示例代码片段为例，介绍这几个函数的使用：

```
/* create and run an animation */
static int do_animation (HWND hWnd)
{
        MGEFF_ANIMATION animation;
        int animation_id = 1;

        int duration;
        int start_val;
        int end_val;

        /* set value */
        duration = DURATION;
        start_val = START_VAL;
        end_val = END_VAL;

        /* create animation object */
        animation = mGEffAnimationCreate ((void *) hWnd, (void *) prop

        /* set animation property */
        /* duration */
        mGEffAnimationSetDuration (animation, duration);

        /* start value */
        mGEffAnimationSetStartValue (animation, &start_val);

        /* end value */
        mGEffAnimationSetEndValue (animation, &end_val);

        /* running */
```

```
        mGEffAnimationSyncRun (animation);

        /* delete the animation object */
        mGEffAnimationDelete (animation);

        return 0;
}
```

上述代码中，startvalue和endvalue是RGB颜色中红色值的起始值和结束值，改动画运行的效果就是将红色值从0x00变化到0xFF。

运行结果

本章小结

本章介绍了动画的几项运行时的参数，包括属性起止值和持续时间，mGEff会在动画运行过程中将属性值按照一定的变化规律（动画曲线），从起始值变化到结束值。并将变化值通过帧回调函数通知给用户，用户可以根据这些绘制动画的每一帧。

-- XuBinWang - 12 Jan 2011

动力曲线

动力曲线

前面的章节介绍了如何创建及运行一个动画，通过执行程序得到的控制台输出可以看到从起始值到终止值的变化过程。但同时也可以看到属性值在起止值之间的变化是均匀的，即为线性变化。如果将这种变化应用到实际动画得到的效果会比较直接和呆板，而用户往往需要将属性值进行诸如加速、减速甚至振荡等非线性的变化，以便达到更灵活真实的动画特效，这就需要用到 mGEff 中的另一个特性——“动力曲线”，本章介绍的及时如何通过它控制动画的运行，以达到属性值的不同变化效果。

函数原型

mGEff提供了一个函数用于设置动力曲线，其原型如下：

设置动力曲线函数

```
void  mGEffAnimationSetCurve(MGEFF_ANIMATION handle, enum EffMotionTyp
```

- 参数说明：
  - handle - 要设置动力曲线的动画句柄；
  - type - 预置曲线的类型，类型见下一节；
- 返回值说明：
  - void - 无返回值；

预置的动力曲线类型

mGEff 中预置了几十种动力曲线以便用户选用，通过给动画设置这些不同的曲线可以达到各种变化效果：

```
enum EffMotionType {
        Linear, /* 线性曲线，动画创建后默认的曲线类型 */
        InQuad, OutQuad, InOutQuad, OutInQuad,
        InCubic, OutCubic, InOutCubic, OutInCubic,
        InQuart, OutQuart, InOutQuart, OutInQuart,
        InQuint, OutQuint, InOutQuint, OutInQuint,
        InSine, OutSine, InOutSine, OutInSine,
```

```
        InExpo, OutExpo, InOutExpo, OutInExpo,
        InCirc, OutCirc, InOutCirc, OutInCirc,
        InElastic, OutElastic, InOutElastic, OutInElastic,
        InBack, OutBack, InOutBack, OutInBack,
        InBounce, OutBounce, InOutBounce, OutInBounce,
        InCurve, OutCurve, SineCurve, CosineCurve,
        Custom, NCurveTypes
};
```

曲线示例

在这里我们对之前的示例代码进行扩充，其值按照InQuad曲线方式变化：

```
/* create and run an animation */
static int do_animation (HWND hWnd)
{
        MGEFF_ANIMATION animation;
        int animation_id = 1;

        int duration;
        int start_val;
        int end_val;

        /* set value */
        duration = DURATION;
        start_val = START_VAL;
        end_val = END_VAL;

        /* create animation object */
        animation = mGEffAnimationCreate ((void *) hWnd, (void *) prop

        /* set animation property */
        /* duration */
        mGEffAnimationSetDuration (animation, duration);

        /* start value */
        mGEffAnimationSetStartValue (animation, &start_val);

        /* end value */
        mGEffAnimationSetEndValue (animation, &end_val);

        /* set curve */
        mGEffAnimationSetCurve (animation, InQuad);

        /* running */
        mGEffAnimationSyncRun (animation);
```

```
        /* delete the animation object */
        mGEffAnimationDelete (animation);

        return 0;
}
```

从执行结果可以看到改变动画的动力曲线后，与之前默认的线性变化输出的插值有了明显的区别：属性值呈加速规律变化。可以想像如果将 printf 语句替换为真正设置对象属性的代码后（如修改圆的半径，矩形的宽高等），使动画带有变速效果。

运行结果

本章小结

本章介绍了动力曲线的作用及预置曲线的使用方法，不同的曲线会给动画带来不同的变化效果，使动画更加真实和灵活。在后面的章节中，还会介绍自定义曲线的使用方法。

-- XuBinWang - 12 Jan 2011

r11 - 14 Feb 2011 - 11:14:36 - XuBinWang

动画的属性

动画的属性

在开始这一章之前，需要先明确两个概念”属性动画“和”动画的属性“：

属性动画，指的是通过修改动画中某个目标对象的某些属性，比如大小，颜色等来达到目标对象动态变化的一种动画；

动画的属性，指的是这个动画自身的一些属性，比如前一章所说的运行时间，总的运行次数，当前运行次数等之类的属性。

两者不可混淆。

函数原型

mGEff 提供了如下函数来设置和获取动画的属性，其原型如下：

设置动画属性函数

```
void mGEffAnimationSetProperty(MGEFF_ANIMATION handle, enum EffAnimPro
```

- 参数说明：
  - handle - 动画对象句柄；
  - id - 属性的标志，具体值可参考 EffAnimProperty? 枚举定义；
  - value - 要设置的属性值；
- 返回值说明：
  - void - 无返回值；

获取动画属性函数

```
int mGEffAnimationGetProperty(MGEFF_ANIMATION handle, enum EffAnimProp
```

- 参数说明：
  - handle - 动画对象句柄；
  - id - 属性的标志，具体值可参考 EffAnimProperty? 枚举定义；
- 返回值说明：

- int - 当前属性值；

mGEff 支持的动画属性

目前 mGEff 支持的动画属性定义如下：

```
enum EffAnimProperty {
        MGEFF_PROP_DURATION    = 0,     /* 可读写属性，动画执行一次的持续时间
        MGEFF_PROP_CURLOOP     = 1,     /* 只读属性，动画当前循环运行的次数  *
        MGEFF_PROP_LOOPCOUNT   = 2,     /* 可读写属性，动画循环运行的次数    *
        MGEFF_PROP_DIRECTION   = 3,     /* 可读写属性，动画运行的方向       */
        MGEFF_PROP_CURFRAME    = 4,     /* 只读属性，当前帧                */
        MGEFF_PROP_STATE       = 5,     /* 只读属性，当前状态               */
        MGEFF_PROP_MAX
};
```

请注意上述属性的注释，里面指明了哪些属性的只读属性，哪些属性是可以设置的，在使用过程应该区分开来。

使用示例

下面通过扩展之前的示例来演示如何设置和获取动画的属性：

```
/* callback function called when property change */
static void property_callback (MGEFF_ANIMATION handle, HWND hWnd, int
{
        /* set animation property */
        g_value = *value;

        /* get current loop */
        g_loop = mGEffAnimationGetProperty (handle, MGEFF_PROP_CURLOOP

        /* update */
        InvalidateRect (hWnd, NULL, TRUE);
}

/* create and run an animation */
static int do_animation (HWND hWnd)
{
        MGEFF_ANIMATION animation;
        int animation_id = 1;

        int duration;
        int start_val;
        int end_val;

        /* set value */
        duration = DURATION;
        start_val = START_VAL;
```

```
        end_val = END_VAL;

        /* create animation object */
        animation = mGEffAnimationCreate ((void *) hWnd, (void *) prop

        /* set animation property */
        /* set loopcount */
        mGEffAnimationSetProperty (animation, MGEFF_PROP_LOOPCOUNT, LO

        /* duration */
        mGEffAnimationSetDuration (animation, duration);

        /* start value */
        mGEffAnimationSetStartValue (animation, &start_val);

        /* end value */
        mGEffAnimationSetEndValue (animation, &end_val);

        /* running */
        mGEffAnimationSyncRun (animation);

        /* delete the animation object */
        mGEffAnimationDelete (animation);

        return 0;
}
```

上例中在创建动画时指定了动画的 MGEFF_PROP_LOOPCOUNT 属性为 3，即动画运行后将循环 3 次。并在属性变化回调函数里获得当前循环过的次数。

运行结果

本章小结

本章主要介绍了“属性动画”与“动画属性”两个概念的区别，并针对“动画属性”的设置与获取进行了介绍。通过动画的属性，可以很方便的操作动画的一些行为。

-- XuBinWang - 12 Jan 2011

动画上下文

动画上下文

一个动画可以设置上下文以便携带更多的用户信息，该特性常被用在属性变化回调函数里面。

函数原型

mGEff提供了如下函数用来设置和获取动画的上下文。其原型如下：

设置动画上下文函数

```
void  mGEffAnimationSetContext(MGEFF_ANIMATION handle, void* context);
```

- 参数说明：
  - handle - 动画对象句柄；
  - context - 上下文信息指针；
- 返回值说明：
  - void - 无返回值；

获取动画上下文函数

```
void *mGEffAnimationGetContext(MGEFF_ANIMATION handle);
```

- 参数说明：
  - handle - 从句柄为 handle 的动画中获取上下文信息；
- 返回值说明：
  - context - 动画对象的上下文信息指针；

通过上面两个函数，可以设置和获得动画的上下文。上下文指针都是 void* 型的，表明可以接收任意类型的指针，其具体的类型由用户约定，这里需要注意一个生命域的问题，下面将有一节专门讲解这个。

使用示例

同样以之前的示例片断为例来介绍这一特性的使用方法：

```
/* callback of property */
static void property_callback (MGEFF_ANIMATION handle, HWND hWnd, int
{
        MY_CONTEXT *pContext;

        /* get and print context */
        pContext = (MY_CONTEXT *)mGEffAnimationGetContext (handle);

        if (pContext->value != *value) {
                pContext->value = *value;
                printf ("CONTEXT: name(%s) value(%d)\n", pContext->nam
        }

        /* set animation property */
        g_value = *value;

        /* get current loop */
        g_loop = mGEffAnimationGetProperty (handle, MGEFF_PROP_CURLOOP

        /* update */
        InvalidateRect (hWnd, NULL, TRUE);
}

/* create and run an animation */
static int do_animation (HWND hWnd)
{
        MGEFF_ANIMATION animation;
        int animation_id = 1;

        int duration;
        int start_val;
        int end_val;

        /* set value */
        duration  = DURATION;
        start_val = START_VAL;
        end_val   = END_VAL;

        /* create animation object */
        animation = mGEffAnimationCreate ((void *) hWnd, (void *) prop

        /* set animation property */
        /* set loopcount */
        mGEffAnimationSetProperty (animation, MGEFF_PROP_LOOPCOUNT, LO

        /* duration */
```

```
        mGEffAnimationSetDuration (animation, duration);

        /* start value */
        mGEffAnimationSetStartValue (animation, &start_val);

        /* end value */
        mGEffAnimationSetEndValue (animation, &end_val);

        /* set context */
        mGEffAnimationSetContext (animation, (void *)&g_my_context);

        /* running */
        mGEffAnimationSyncRun (animation);

        /* delete the animation object */
        mGEffAnimationDelete (animation);

        return 0;
}
```

运行结果

上下文生命域

由于动画上下文只是一个 void* 型的指针，所以使用时具体类型要由用户来约定，如果用户为上下文动态分配了空间，也应该自己在动画完成后将其释放。另外需要注意的是如果上下文是在栈上分配的，而动画以异步方式运行（详情请参考《动画运行方式》一章），那么很可能在动画的属性变化回调被调用时已经离开了上下文的生命域，此时获得到的上下文指针已经指向无效的内容，将会导致不可预料的运行错误，例如将上例中的 mGEffAnimationSyncRun 函数替换为 mGEffAnimationAsyncRun，就可能会出现这种情况，因为 context 数组是在栈上的。要避免这种情况发生有多种方法，比如可以在离开 context 生命域之前调用 mGEffAnimationWait 函数等待异步动画结束，或者在堆上分配 context 空间，并在动画结束时释放（至于如何掌握动画的结束时机，请参考《回调函数》章节）。

本章小结

本章介绍了动画上下文这一特性及其应用方式，通过上下文要以给动画附加更多信息并在有需要的地方获取这些信息。不过用户需要保证 context 的生命周期不能短于动画生命周期，以防止出现在动画执行中获取不到正确上下文的情况。

-- XuBinWang - 12 Jan 2011

动画的运行方式及控制

动画的运行方式

动画支持同步及异步两种不同的执行方式。采用同步方式运行的动画，程序阻塞在运行函数上，直到动画结合，函数返回，程序才继续运行，在这个过程中程序将无法响应消息；而采用异步方式运行的动画，运行函数调用后立即返回，此时动画在后台被调度执行，直到被中止或结束，而程序可以相应消息处理。下面将分别介绍这两种运行方式的使用和控制。

函数原型

截止到本章为止，我们在所有的示例程序中的动画都是以同步方式运行的，同步动画的使用比较简单，直接在动画创建后调用 mGEffAnimationSyncRun 函数就可以开始动画并且直到其结束才返回，整个过程是阻塞的，在此期间程序是不能响应消息处理。接下我们开始介绍异步运行动画函数和如何控制动画的运行。

mGEff 提供了以下几个函数来运行和控制动画。其原型分别为：

同步运行动画函数

```
int  mGEffAnimationSyncRun(MGEFF_ANIMATION handle);
```

- 参数说明：
  - handle - 动画对象句柄；
- 返回值说明：
  - int - 0；

异步运行动画函数

```
int  mGEffAnimationAsyncRun(MGEFF_ANIMATION handle);
```

- 参数说明：
  - handle - 动画对象句柄；
- 返回值说明： *int - 0；

等待异步动画运行结束函数

```
MGEFF_BOOL mGEffAnimationWait(void* phWnd, MGEFF_ANIMATION handle);
```

- 参数说明：
  - phWnd - 当前线程的窗体句柄的地址；
  - handle - 动画对象句柄；
- 返回值说明：
  - MGEFF_BOOL - 成功返回 MGEFF_TRUE，否则返回 MGEFF_FALSE

暂停动画运行函数

```
void mGEffAnimationPause(MGEFF_ANIMATION handle);
```

- 参数说明： * handle - 动画对象句柄；
- 返回值说明：
  - void - 无返回值；

恢复动画运行函数

```
void mGEffAnimationResume(MGEFF_ANIMATION handle);
```

- 参数说明： * handle - 动画对象句柄；
- 返回值说明：
  - void - 无返回值；

停止动画运行函数

```
void mGEffAnimationStop(MGEFF_ANIMATION handle);
```

- 参数说明： * handle - 动画对象句柄；
- 返回值说明：
  - void - 无返回值；

使用示例

```
#include <stdio.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/gdi.h>
```

```
#include <minigui/window.h>
#include <minigui/minigui.h>

#include <mgeff/mgeff.h>

/*********************************************************************
#define CAPTION    "animation_control"
#define BAR_HEIGHT 50
#define DURATION   (20 * 1000)
#define START_VAL  0x00
#define END_VAL    0xFF

/*********************************************************************
static char g_str[64];
static int g_value = 0x00;
static MGEFF_ANIMATION g_handle = NULL;

/*********************************************************************
/* main window proc */
static int mainWindowProc (HWND hWnd, int message, WPARAM wParam, LPAR
/* draw a frame */
static void draw_frame (HWND hWnd);
/* callback function called when property change */
static void property_callback (MGEFF_ANIMATION handle, HWND hWnd, int
/* create and run an animation */
static int do_animation (HWND hWnd);

/*********************************************************************
int MiniGUIMain (int argc, const char *argv[])
{
        HWND hMainHwnd;
        MAINWINCREATE createInfo;
        MSG msg;

        #ifdef _MGRM_PROCESSES
        JoinLayer (NAME_DEF_LAYER, "animation", 0, 0);
        #endif

        createInfo.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
        createInfo.dwExStyle = WS_EX_NONE;
        createInfo.spCaption = CAPTION;
        createInfo.hMenu = 0;
        createInfo.hCursor = GetSystemCursor (0);
        createInfo.hIcon = 0;
        createInfo.MainWindowProc = mainWindowProc;
        createInfo.lx = 0;
        createInfo.ty = 0;
        createInfo.rx = 240;
```

```
        createInfo.by = 320;
        createInfo.iBkColor = COLOR_lightwhite;
        createInfo.dwAddData = 0;
        createInfo.hHosting = HWND_DESKTOP;

        hMainHwnd = CreateMainWindow (&createInfo);

        if (hMainHwnd == HWND_INVALID) {
                return -1;
        }

        ShowWindow (hMainHwnd, SW_SHOWNORMAL);

        while (GetMessage (&msg, hMainHwnd)) {
                TranslateMessage (&msg);
                DispatchMessage (&msg);
        }

        MainWindowThreadCleanup (hMainHwnd);

        return 0;
}

/************************************************************************
/* main window proc */
static int mainWindowProc (HWND hWnd, int message, WPARAM wParam, LPAR
{
        switch (message) {
                case MSG_CREATE:
                /* init mGEff library */
                mGEffInit ();

                sprintf (g_str, "you can press any key.");

                do_animation (hWnd);
                break;

                case MSG_KEYDOWN:
                sprintf (g_str, "you press key, keycode(%d)", wParam);

                printf ("%s\n", g_str);

                if (g_handle != NULL) {
                        if (wParam == SCANCODE_S) {
                                /* stop animation */
                                mGEffAnimationStop (g_handle);
                                sprintf (g_str, "stop animation");
                        }
```

```
                        if (wParam == SCANCODE_P) {
                                /* pause animation */
                                mGEffAnimationPause (g_handle);
                                sprintf (g_str, "pause animation");
                        }

                        if (wParam == SCANCODE_R) {
                                /* resume animation */
                                mGEffAnimationResume (g_handle);
                                sprintf (g_str, "resume animation");
                        }
                }

                InvalidateRect (hWnd, NULL, TRUE);
                break;

                case MSG_PAINT:
                draw_frame (hWnd);
                break;

                case MSG_CLOSE:
                DestroyMainWindow (hWnd);
                PostQuitMessage (hWnd);

                /* deinit mGEff library */
                mGEffDeinit ();
                break;

                default:
                break;
        }

        return DefaultMainWinProc (hWnd, message, wParam, lParam);
}

/* draw a frame */
static void draw_frame (HWND hWnd)
{
        HDC dc;
        RECT rc;

        int client_w, client_h;
        char str[64];
        int color;

        /* begin draw */
        dc = BeginPaint (hWnd);
```

```
        /* get client rect */
        GetClientRect (hWnd, &rc);

        client_w = RECTW (rc);
        client_h = (RECTH (rc) - BAR_HEIGHT) * g_value / (END_VAL - ST

        color = g_value;

        /* set brush and draw area */
        SetBrushColor (dc, RGB2Pixel (dc, color, 0x00, 0x00));

        FillBox (dc, 0, BAR_HEIGHT, client_w, client_h);

        /* draw the text */
        sprintf (str, "current color: RGB(%d,0,0)", color);

        TextOut (dc, 0, 0, str);

        /* print information */
        TextOut (dc, 0, BAR_HEIGHT / 2, g_str);

        EndPaint (hWnd, dc);
        /* end draw */
}

/* callback function called when property change */
static void property_callback (MGEFF_ANIMATION handle, HWND hWnd, int
{
        /* set animation property */
        g_value = *value;

        /* update */
        InvalidateRect (hWnd, NULL, TRUE);
}

/* create and run an animation */
static int do_animation (HWND hWnd)
{
        MGEFF_ANIMATION animation;
        int animation_id = 1;

        int duration;
        int start_val;
        int end_val;

        /* set value */
        duration = DURATION;
```

```
        start_val = START_VAL;
        end_val = END_VAL;

        /* create animation object */
        animation = mGEffAnimationCreate ((void *) hWnd, (void *) prop

        g_handle = animation;

        /* set animation property */
        /* duration */
        mGEffAnimationSetDuration (animation, duration);

        /* start value */
        mGEffAnimationSetStartValue (animation, &start_val);

        /* end value */
        mGEffAnimationSetEndValue (animation, &end_val);

        /* set loopcount */
        //    mGEffAnimationSetProperty (animation, MGEFF_PROP_LOOPCOU

        /* running */
        mGEffAnimationAsyncRun (animation);

        /* wait animation stop */
        mGEffAnimationWait ((void *) &hWnd, animation);

        g_handle = NULL;

        /* delete the animation object */
        mGEffAnimationDelete (animation);

        return 0;
}
```

运行结果

注意事项

上述几个函数禁止用于同步运行动画对象。

本章小结

本章介绍了动画的两种运行方式——同步和异步。这两种方式有各自的应用场景，现将其优缺点整理如下：

- 同步动画

    - 优点
        - 使用简单
        - 易于掌握生命周期
    - 缺点
        - 运行时程序无法响应其它消息
- 异步动画
    - 优点
        - 运行时程序仍然可以响应其它消息
    - 缺点
        - 用户可能需要自行控制其生命周期
        - 涉及的函数较多，使用稍为复杂

-- XuBinWang - 12 Jan 2011

动画组

动画组

在实际应用中，用户往往不只是希望同时执行一个动画，而是需要将多个动画关联起来并行或串行地执行，以获得更复杂的动画效果。所以 mGEff 引入了动画组机制，用户可以通过创建并行 (Parallel) 或串行 (Sequential) 组来达到执行一组动画的需求。

- 并行组动画
  并行组的特点是当动画组执行时组内所有动画都同时被调度，表现的效果是这些动画同时在进行，组动画的总运行时间由最长的那个动画运行时间决定
- 串行组动画
  串行组的特点是动画组执行时组内动画将被顺序调度，表现的效果是这些动画分先后秩序进行，组动画的总运行时间为所有动画运行时间的和。另外可以将一个动画组当作一个动画添加到另一个组，其调度规则同上。

函数原型

mGEff 提供了如下几个函数用于支持动画组的创建和使用。其函数原型如下：

创建动画组函数

```
MGEFF_ANIMATION mGEffAnimationCreateGroup(enum EffAnimationType type);
```

- 参数说明：
  - type - 动画组类型，可以是 MGEFF_PARALLEL （并行）和 MGEFF_SEQUENTIAL （串行）二值之一；
- 返回值说明：
  - handle - 成功返回动画组句柄，否则返回 NULL；

添加到动画组函数

```
/**
* 将一个动画加入动画组
```

```
*/
void mGEffAnimationAddToGroup(MGEFF_ANIMATION group, MGEFF_ANIMATION a
```

- 参数说明：
  - group - 动画组句柄；
  - animation - 将加入 group 中的动画句柄；
- 返回值说明：
  - void - ；

一点概念

从函数原型可以看到，动画组的句柄和动画句柄是同样的类型，前面介绍过 mGEff 的设计上采用了一些面向对象思想，动画组对象从动画对象“继承”而来。动画组也是一种动画，之前介绍过的动画属性、上下文和运行方式相关的函数均可作用于动画组，这一点将在下一节的示例中展示出来。

使用示例

并行动画组

```
#include <stdio.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/minigui.h>

#include <mgeff/mgeff.h>

/*********************************************************************
#define CAPTION    "group_animation_parallel"
#define BAR_HEIGHT 50
#define DURATION   (5 * 1000)
#define START_VAL  0x00
#define END_VAL    0xFF
#define ANIMATION_NUM 3

/*********************************************************************
static int g_color[ANIMATION_NUM] = { 0 };

/*********************************************************************
/* main window proc */
static int mainWindowProc (HWND hWnd, int message, WPARAM wParam, LPAR
/* draw a frame */
static void draw_frame (HWND hWnd);
/* callback function called when property change */
static void property_callback (MGEFF_ANIMATION handle, HWND hWnd, int
```

```
/* create and run an animation */
static int do_animation (HWND hWnd);

/*****************************************************************
int MiniGUIMain (int argc, const char *argv[])
{
        HWND hMainHwnd;
        MAINWINCREATE createInfo;
        MSG msg;

        #ifdef _MGRM_PROCESSES
        JoinLayer (NAME_DEF_LAYER, "animation", 0, 0);
        #endif

        createInfo.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
        createInfo.dwExStyle = WS_EX_NONE;
        createInfo.spCaption = CAPTION;
        createInfo.hMenu = 0;
        createInfo.hCursor = GetSystemCursor (0);
        createInfo.hIcon = 0;
        createInfo.MainWindowProc = mainWindowProc;
        createInfo.lx = 0;
        createInfo.ty = 0;
        createInfo.rx = 240;
        createInfo.by = 320;
        createInfo.iBkColor = COLOR_lightwhite;
        createInfo.dwAddData = 0;
        createInfo.hHosting = HWND_DESKTOP;

        hMainHwnd = CreateMainWindow (&createInfo);

        if (hMainHwnd == HWND_INVALID) {
                return -1;
        }

        ShowWindow (hMainHwnd, SW_SHOWNORMAL);

        while (GetMessage (&msg, hMainHwnd)) {
                TranslateMessage (&msg);
                DispatchMessage (&msg);
        }

        MainWindowThreadCleanup (hMainHwnd);

        return 0;
}

/*****************************************************************
```

```
static int mainWindowProc (HWND hWnd, int message, WPARAM wParam, LPAR
{
        switch (message) {
                case MSG_CREATE:
                /* init mgeff library */
                mGEffInit ();

                do_animation (hWnd);
                break;

                case MSG_PAINT:
                draw_frame (hWnd);
                break;

                case MSG_CLOSE:
                DestroyMainWindow (hWnd);
                PostQuitMessage (hWnd);

                /* deinit mgeff library */
                mGEffDeinit ();
                break;

                default:
                break;
        }

        return DefaultMainWinProc (hWnd, message, wParam, lParam);
}

static void draw_frame (HWND hWnd)
{
        HDC dc;
        RECT rc;

        int client_w, client_h;
        char str[ANIMATION_NUM][64];

        dc = BeginPaint (hWnd);

        /* get client rect */
        GetClientRect (hWnd, &rc);

        client_w = RECTW (rc);
        client_h = RECTH (rc);// - BAR_HEIGHT) * g_value / (END_VAL -

        /* draw */
        /* red */
        SetBrushColor (dc, RGB2Pixel (dc, g_color[0], 0, 0));
```

```
        FillBox (dc, 0, client_h * 0 / ANIMATION_NUM, client_w, client

        sprintf (str[0], "%d", g_color[0]);
        TextOut (dc, 0, client_h * 0 / ANIMATION_NUM, str[0]);

        /* green */
        SetBrushColor (dc, RGB2Pixel (dc, 0, g_color[1], 0));
        FillBox (dc, 0, client_h * 1 / ANIMATION_NUM, client_w, client

        sprintf (str[1], "%d", g_color[1]);
        TextOut (dc, 0, client_h * 1 / ANIMATION_NUM, str[1]);

        /* blue */
        SetBrushColor (dc, RGB2Pixel (dc, 0, 0, g_color[2]));
        FillBox (dc, 0, client_h * 2 / ANIMATION_NUM, client_w, client

        sprintf (str[2], "%d", g_color[2]);
        TextOut (dc, 0, client_h * 2 / ANIMATION_NUM, str[2]);

        EndPaint (hWnd, dc);
}

static void property_callback (MGEFF_ANIMATION handle, HWND hWnd, int
{
        g_color[id] = *value;

        InvalidateRect (hWnd, NULL, TRUE);
}

static int do_animation (HWND hWnd)
{
        MGEFF_ANIMATION animation[ANIMATION_NUM];
        MGEFF_ANIMATION group_animation;

        int duration;;
        int start_val;
        int end_val;

        int i;

        /* set value */
        duration = DURATION;
        start_val = START_VAL;
        end_val = END_VAL;

        /* create an animation group */
        group_animation = mGEffAnimationCreateGroup (MGEFF_PARALLEL);
        //group_animation = mGEffAnimationCreateGroup (MGEFF_SEQUENTIA
```

```
        /* create and animation and add it to a group */
        for (i = 0; i < ANIMATION_NUM; i++) {
                /* create animation */
                animation[i] = mGEffAnimationCreate ((void *) hWnd, (v

                /* set property */
                /* duration */
                mGEffAnimationSetDuration (animation[i], duration);

                /* start value */
                mGEffAnimationSetStartValue (animation[i], &start_val)

                /* end value */
                mGEffAnimationSetEndValue (animation[i], &end_val);

                /* add animation to group */
                mGEffAnimationAddToGroup (group_animation, animation[i
        }

        /* running */
        mGEffAnimationAsyncRun (group_animation);

        /* wait animation end */
        mGEffAnimationWait ((void *) &hWnd, group_animation);

        /* delete the animation object */
        mGEffAnimationDelete (group_animation);

        return 0;
}
```

串行动画组

只要将上个示例中的创建动画组函数调用替换成下面这句，就可以实现一个串行动画组：

```
/* ... */
group = mGEffAnimationCreateGroup(MGEFF_SEQUENTIAL);
/* ... */
```

本章小结

-- XuBinWang - 12 Jan 2011

动画回调函数

回调函数

在 mGEff 的动画播放或其状态发生改变时，框架会尝试调用用户设置的回调函数，以便让用户做出一些相应的处理，下面介绍一下这些回调函数：

函数原型

动画结束时的回调函数

如果用户设置了动画结束回调函数，将在动画播放完后被调用，一般可以用来在动画结束时释放资源。具体回调函数原型和设置回调的 API 如下：

```
/* 动画结束时调用的回调原型 */
typedef void (*MGEFF_FINISHED_CB)(MGEFF_ANIMATION handle);
```

- 参数说明：
  - handle - 动画对象句柄；
- 返回值说明：
  - void - 无返回值；

```
/* 设置动画结束时要调用的回调函数 */
void mGEffAnimationSetFinishedCb(MGEFF_ANIMATION handle, MGEFF_FINISHE
```

- 参数说明：
  - handle - 动画对象句柄；
  - cb - 回调函数指针；
- 返回值说明：
  - void - 无返回值；

动画循环次数改变时的回调函数

当动画循环次数发生改变，而且用户设置了相应的回调函数时，mGEff 会调用这个回调函数：

```
/* 动画循环次数发生改变时调用的回调原型 */
typedef void (*MGEFF_CURLOOPCHANGED_CB)(MGEFF_ANIMATION handle);
```

- 参数说明：
  - handle - 动画对象句柄；
- 返回值说明：
  - void - 无返回值；

```
/* 设置动画循环次数改变时要调用的回调函数 */
void mGEffAnimationSetCurLoopChangedCb(MGEFF_ANIMATION handle, MGEFF_C
```

- 参数说明：
  - handle - 动画对象句柄；
  - cb - 回调函数指针；
- 返回值说明：
  - void - 无返回值；

动画运行方向改变时的回调函数

当动画的运行方向发生改变，而且用户设置了相应的回调函数时，该回调将被 mGEff 调用：

```
/* 动画运行方向发生改变时调用的回调原型 */
typedef void (*MGEFF_DIRCHANGED_CB)(MGEFF_ANIMATION handle);
```

- 参数说明：
  - handle - 动画对象句柄；
- 返回值说明：
  - void - 无返回值；

```
/* 设置动画运行方向改变时要调用的回调函数 */
void mGEffAnimationSetDirChangedCb(MGEFF_ANIMATION handle, MGEFF_DIRCH
```

- 参数说明：
  - handle - 动画对象句柄；
  - cb - 回调函数指针；
- 返回值说明：
  - void - 无返回值；

动画状态改变时的回调函数

当动画的状态发生改变，而且用户设置了相应的回调函数时，该回调将被 mGEff 调用：

```
/* 动画状态发生改变时调用的回调原型 */
typedef void (*MGEFF_STATECHANGED_CB)(MGEFF_ANIMATION handle, enum Eff
```

- 参数说明：

    - handle - 动画对象句柄；
    - newEffState - 新的状态值；
    - oldEffState - 旧的状态值；
- 返回值说明：
    - void - 无返回值；

```
/* 设置动画状态改变时要调用的回调函数 */
void mGEffAnimationSetStateChangedCb(MGEFF_ANIMATION handle, MGEFF_STA
```

- 参数说明：
    - handle - 动画对象句柄；
    - cb - 回调函数指针；
- 返回值说明：
    - void - 无返回值；

使用示例

```
#include <stdio.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/minigui.h>

#include <mgeff/mgeff.h>

/*********************************************************************
#define CAPTION    "animation_callback"
#define BAR_HEIGHT 50
#define DURATION   (20 * 1000)
#define LOOPCOUNT  3
#define START_VAL  0x00
#define END_VAL    0xFF

/*********************************************************************
static char g_str[64];
static int g_value = 0x00;
static int g_loop = 0;

/*********************************************************************
/* main window proc */
static int mainWindowProc (HWND hWnd, int message, WPARAM wParam, LPAR
/* draw a frame */
static void draw_frame (HWND hWnd);
/* callback function called when property change */
```

```
static void property_callback (MGEFF_ANIMATION handle, HWND hWnd, int
/* create and run an animation */
static int do_animation (HWND hWnd);

/* callback function called when animation finished */
static void finished_callback (MGEFF_ANIMATION handle);
/* callback function called when animation loop count is changed */
static void curloopchanged_callback (MGEFF_ANIMATION handle);
/* callback function called when animation direction changed */
static void dirchanged_callback (MGEFF_ANIMATION handle);
/* callback function called when animation state changed */
static void statechanged_callback (MGEFF_ANIMATION handle, enum EffSta

/*********************************************************************
int MiniGUIMain (int argc, const char *argv[])
{
        HWND hMainHwnd;
        MAINWINCREATE createInfo;
        MSG msg;

        #ifdef _MGRM_PROCESSES
        JoinLayer (NAME_DEF_LAYER, "animation", 0, 0);
        #endif

        createInfo.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
        createInfo.dwExStyle = WS_EX_NONE;
        createInfo.spCaption = CAPTION;
        createInfo.hMenu = 0;
        createInfo.hCursor = GetSystemCursor (0);
        createInfo.hIcon = 0;
        createInfo.MainWindowProc = mainWindowProc;
        createInfo.lx = 0;
        createInfo.ty = 0;
        createInfo.rx = 240;
        createInfo.by = 320;
        createInfo.iBkColor = COLOR_lightwhite;
        createInfo.dwAddData = 0;
        createInfo.hHosting = HWND_DESKTOP;

        hMainHwnd = CreateMainWindow (&createInfo);

        if (hMainHwnd == HWND_INVALID) {
                return -1;
        }

        ShowWindow (hMainHwnd, SW_SHOWNORMAL);

        while (GetMessage (&msg, hMainHwnd)) {
```

```
                TranslateMessage (&msg);
                DispatchMessage (&msg);
        }

        MainWindowThreadCleanup (hMainHwnd);

        return 0;
}

/*************************************************************
/* main window proc */
static int mainWindowProc (HWND hWnd, int message, WPARAM wParam, LPAR
{
        switch (message) {
                case MSG_CREATE:
                /* init mGEff library */
                mGEffInit ();

                sprintf (g_str, "you can press any key.");

                do_animation (hWnd);
                break;

                case MSG_KEYDOWN:
                sprintf (g_str, "you press key, keycode(%d)", wParam);

                printf ("%s\n", g_str);

                InvalidateRect (hWnd, NULL, TRUE);
                break;

                case MSG_PAINT:
                draw_frame (hWnd);
                break;

                case MSG_CLOSE:
                DestroyMainWindow (hWnd);
                PostQuitMessage (hWnd);

                /* deinit mGEff library */
                mGEffDeinit ();
                break;

                default:
                break;
        }

        return DefaultMainWinProc (hWnd, message, wParam, lParam);
```

```
}

/* draw a frame */
static void draw_frame (HWND hWnd)
{
        HDC dc;
        RECT rc;

        int client_w, client_h;
        char str[64];
        int color;

        /* begin draw */
        dc = BeginPaint (hWnd);

        /* get client rect */
        GetClientRect (hWnd, &rc);

        client_w = RECTW (rc);
        client_h = (RECTH (rc) - BAR_HEIGHT) * g_value / (END_VAL - ST

        color = g_value;

        /* set brush and draw area */
        SetBrushColor (dc, RGB2Pixel (dc, color, 0x00, 0x00));

        FillBox (dc, 0, BAR_HEIGHT, client_w, client_h);

        /* draw the text */
        sprintf (str, "current color: RGB(%d,0,0)", color);

        TextOut (dc, 0, 0, str);

        /* print information */
        TextOut (dc, 0, BAR_HEIGHT / 3, g_str);

        /* print curloop */
        sprintf (str, "current loop: (%d)", g_loop);

        TextOut (dc, 0, 2 * BAR_HEIGHT / 3, str);

        EndPaint (hWnd, dc);
        /* end draw */
}

/* callback function called when property change */
static void property_callback (MGEFF_ANIMATION handle, HWND hWnd, int
{
```

```
        /* set animation property */
        g_value = *value;

        /* get current loop */
        g_loop = mGEffAnimationGetProperty (handle, MGEFF_PROP_CURLOOP

        /* update */
        InvalidateRect (hWnd, NULL, TRUE);
}

/* create and run an animation */
static int do_animation (HWND hWnd)
{
        MGEFF_ANIMATION animation;
        int animation_id = 1;

        int duration;
        int start_val;
        int end_val;

        /* set value */
        duration = DURATION;
        start_val = START_VAL;
        end_val = END_VAL;

        /* create animation object */
        animation = mGEffAnimationCreate ((void *) hWnd, (void *) prop

        /* set animation property */
        /* set loopcount */
        mGEffAnimationSetProperty (animation, MGEFF_PROP_LOOPCOUNT, LO

        /* duration */
        mGEffAnimationSetDuration (animation, duration);

        /* start value */
        mGEffAnimationSetStartValue (animation, &start_val);

        /* end value */
        mGEffAnimationSetEndValue (animation, &end_val);

        /* set callback */
        mGEffAnimationSetFinishedCb (animation, finished_callback);
        mGEffAnimationSetCurLoopChangedCb (animation, curloopchanged_c
        mGEffAnimationSetDirChangedCb (animation, dirchanged_callback)
        mGEffAnimationSetStateChangedCb (animation, statechanged_callb

        /* running */
```

```
        mGEffAnimationAsyncRun (animation);

        /* wait animation stop */
        mGEffAnimationWait ((void *) &hWnd, animation);

        /* delete the animation object */
        mGEffAnimationDelete (animation);

        return 0;
}

/* callback function called when animation finished */
static void finished_callback (MGEFF_ANIMATION handle)
{
        printf ("finished_callback is called.\n");
}

/* callback function called when animation loop count is changed */
static void curloopchanged_callback (MGEFF_ANIMATION handle)
{
        int cur_loop;

        printf ("curloopchanged_callback is called.\n");

        cur_loop = mGEffAnimationGetProperty (handle, MGEFF_PROP_CURLO

        printf ("current loop : %d\n", cur_loop);
}

/* callback function called when animation direction changed */
static void dirchanged_callback (MGEFF_ANIMATION handle)
{
        int direction;

        printf ("dirchanged_callback is called.\n");

        direction = mGEffAnimationGetProperty (handle, MGEFF_PROP_DIRE

        printf ("current direction : %d\n", direction);
}

/* callback function called when animation state changed */
static void statechanged_callback (MGEFF_ANIMATION handle, enum EffSta
{
        int state;

        printf ("statechanged_callback is called.\n");
```

```
        state = mGEffAnimationGetProperty (handle, MGEFF_PROP_STATE);

        printf ("current state : %d\n", state);
        printf ("new     state : %d\n", newEffState);
        printf ("old     state : %d\n", oldEffState);
}
```

运行结果

```
statechanged_callback is called.
current state : 0
new     state : 3
old     state : 0
statechanged_callback is called.
current state : 3
new     state : 2
old     state : 3
curloopchanged_callback is called.
current loop : 1
curloopchanged_callback is called.
current loop : 2
curloopchanged_callback is called.
current loop : 3
statechanged_callback is called.
current state : 2
new     state : 0
old     state : 2
finished_callback is called.
```

本章小结

本章介绍了 mGEff 框架中的一些回调函数，通过设置这些回调函数，可以在动画的各个阶段以及状态发生改变时被框架所调用，这就可以让用户在动画的不同阶段做出不同的处理。

-- XuBinWang - 12 Jan 2011

特效器

特效器

由于 mGEff 提供的只是动画框架，具体的动画交给用户去实现。而有些动画特效比较通用，如果每次都要让用户实现，即不利于代码复用又无法保证其实现是最优的。所以 mGEff 中引入了 Effector 的概念。简单的讲，Effector 将常用的动画特效（如卷页）实现并封装起来，以便用户直接使用。

特效器的本质是封装特定的动画特效，而在使用方式上它有一个约定：用户创建特效器后传入至少两个素材源（少数特效器需要两个以上素材源），并设置输出目标，这样在特效器开始运行后会将第一个素材源在指定时间内变化为第二个素材源，并将这个过程显示在输出目标上。在当前版本的 mGEff 中素材源支持从 MiniGUI 的 DC 上创建，而输出目标则是 MiniGUI 窗口（或控件）。

函数原型

mGEff 提供了如下函数操作特效器的创建和运行。其原型如下：

创建特效器函数

```
MGEFF_EFFECTOR mGEffEffectorCreate(unsigned long key);
```

- 参数说明：
  - key - 特效器名称的 hash 值，通过该值作为索引可以对应一个唯一的特效器
- 返回值说明：
  - MGEFF_EFFECTOR - 创建成功返回特效器句柄，失败返回 NULL

删除特效器函数

```
void mGEffEffectorDelete(MGEFF_EFFECTOR handle);
```

- 参数说明：
  - handle - 要删除特效器的句柄
- 返回值说明：

通过指定 DC 创建一个特效素材源函数

```
MGEFF_SOURCE mGEffCreateSource(HDC hdc);
```

- 参数说明：
  - hdc - 被用于创建素材源的 DC 句柄
- 返回值说明：
  - MGEFF_SOURCE - 成功返回素材源的句柄，否则返回 NULL

往指定特效器里追加一个特效素材源函数

```
int mGEffEffectorAppendSource(MGEFF_EFFECTOR effector, MGEFF_SOURCE so
```

- 参数说明：
  - effector - 要追加素材的特效器
  - source - 特效素材源句柄
- 返回值说明：
  - int - 该函数目前始终返回 0

创建特效器输出目标函数

```
MGEFF_SINK mGEffCreateHwndSink(HWND hwnd);
```

- 参数说明：
  - hwnd - 用来输出的目标窗口
- 返回值说明：
  - MGEFF_SINK - 成功返回输出目标管道句柄

设置特效器的输出目标

```
int mGEffEffectorSetSink(MGEFF_EFFECTOR effector, MGEFF_SINK sink);
```

- 参数说明：
  - effector - 准备输出的特效器
  - sink - 输出目标
- 返回值说明：
  - int - 该函数目前始终返回 0

从特效器中创建动画

```
MGEFF_ANIMATION mGEffAnimationCreateWithEffector(MGEFF_EFFECTOR effect
```

- 参数说明：
  - effector - 特效器句柄
- 返回值说明：
  - handle - 成功返回动画句柄，否则返回 NULL

内置的特效器

上面的这几个函数可以用来创建及使用特效器，mGEff 提供了一组内置特效器可用来实现一些常见特效，并将这些特效器的 hash 值计算封装成一组宏，可以直接作为参数传入 mGEffEffectorCreate 函数：

```
/* 百叶窗特效器 */
#define MGEFF_EFFECTOR_LEAFWINDOW   mGEffStr2Key(MGEFF_MINOR_leafwindo

/* 压扁特效器 */
#define MGEFF_EFFECTOR_ZIP          mGEffStr2Key(MGEFF_MINOR_zip)

/* 水平翻转特效器 */
#define MGEFF_EFFECTOR_FLIP         mGEffStr2Key(MGEFF_MINOR_flip)

/* 淡出淡入特效器 */
#define MGEFF_EFFECTOR_ALPHA        mGEffStr2Key(MGEFF_MINOR_alpha)

/* 滚屏特效器 */
#define MGEFF_EFFECTOR_SCROLL       mGEffStr2Key(MGEFF_MINOR_scroll)

/* 缩放特效器 */
#define MGEFF_EFFECTOR_ZOOM         mGEffStr2Key(MGEFF_MINOR_zoom)

/* 平推特效器 */
#define MGEFF_EFFECTOR_PUSH         mGEffStr2Key(MGEFF_MINOR_push)

/* 清屏特效器 */
#define MGEFF_EFFECTOR_CLEAR        mGEffStr2Key(MGEFF_MINOR_clear)

/* 立方体旋转特效器 */
#define MGEFF_EFFECTOR_CUBIC_ROTATE mGEffStr2Key(MGEFF_MINOR_cubicrota

/* 中心分裂特效器 */
#define MGEFF_EFFECTOR_CENTERSPLIT  mGEffStr2Key(MGEFF_MINOR_centerspl

/* 雷达特效器 */
#define MGEFF_EFFECTOR_RADARSCAN    mGEffStr2Key(MGEFF_MINOR_radarscan

/* 卷页特效器 */
```

```
#define MGEFF_EFFECTOR_ROLLER       mGEffStr2Key(MGEFF_MINOR_roller)

/* 块翻转特效器 */
#define MGEFF_EFFECTOR_BLOCKFLIP    mGEffStr2Key(MGEFF_MINOR_blockflip

/* 五角星旋转特效器 */
#define MGEFF_EFFECTOR_FIVEPOINTEDSTAR      mGEffStr2Key(MGEFF_MINOR_f

/* 基于 mgplus 的旋转特效器 */
#define MGEFF_EFFECTOR_MGPLUS_ROTATE        mGEffStr2Key(MGEFF_MINOR_m

/* 基于 mgplus 的立方体旋转特效器 */
#define MGEFF_EFFECTOR_MGPLUS_CUBIC_ROTATE  mGEffStr2Key(MGEFF_MINOR_m

/* 基于 mgplus 的水平翻转特效器 */
#define MGEFF_EFFECTOR_MGPLUS_FLIP          mGEffStr2Key(MGEFF_MINOR_m

/* 基于 OpenGL ES 的立方体旋转特效器 */
#define MGEFF_EFFECTOR_OPENGL_CUBICROTATE   mGEffStr2Key(MGEFF_MINOR_e

/* 基于 OpenGL ES 的平面旋转特效器 */
#define MGEFF_EFFECTOR_OPENGL_RECTROTATE    mGEffStr2Key(MGEFF_MINOR_e

/* 基于 OpenGL ES 的 CoverFlow 特效器 */
#define MGEFF_EFFECTOR_OPENGL_COVERFLOW     mGEffStr2Key(MGEFF_MINOR_e
```

内置特效器分类

从上面的内置特效器宏定义可以看到，某些特效器分别有原生版及基于 mgplus 和 OpenGL? ES 的实现版本。这三个版本的特点分别是：

- 原生版速度最优，是经过特别优化适合应用于中低端硬件平台；
- 基于 mgplus 的版本则效果最好，适用于中高端硬件平台；
- 基于 OpenGL? ES 的版本则采用了硬件的 3D 加速支持，适用于提供了这一特性的硬件平台；

值得注意的是 mGEff 默认是不开启对 OpenGL? ES 支持的，要使用这一套特效器，必须在编译时打开相应的编译参数。

特效器的属性

大部分特效器都拥有各自的属性，以便用户对其产生的特效进行调整和一定程序上的定制。这里将 mGEff 目前支持的特效器属性列举出来：

```
/* 方向属性，用来指定特效效果的进行方向，如设置平推特效从右向左推移 */
MGEFF_PROPERTY_DIRECTION

/* 旋转坐标轴属性，用来指定旋转（翻转）类特效的旋转轴，如设置立方体特效绕 Y 轴旋转
```

```
MGEFF_PROPERTY_AXIS

/* 百叶窗特效器的叶片（行）数 */
MGEFF_PROPERTY_LEAFROWS

/* 百叶窗特效器的方向，通过该属性指定是否为垂直方向 */
MGEFF_PROPERTY_LEAFVERTICAL

/* 缩放特效的方向，如设置为从右下向左上角放大 */
MGEFF_PROPERTY_ZOOM

// TODO: 待续
MGEFF_PROPERTY_RESOURCE
MGEFF_PROPERTY_BACKGROUND
MGEFF_PROPERTY_PIECES
MGEFF_PROPERTY_STARTANGLE
```

使用示例

```
#include <stdio.h>
#include <assert.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/minigui.h>

#include <mgeff/mgeff.h>

static HDC createDCByPicture(HDC hdc, int color)
{
        HDC dc;
        int w,h;

        w = GetGDCapability(hdc, GDCAP_MAXX) + 1;
        h = GetGDCapability(hdc, GDCAP_MAXY) + 1;
        dc = CreateCompatibleDCEx(hdc, w, h);
        SetBrushColor(dc, color);
        FillBox(dc, 0, 0, w, h);

        return dc;
}

int fillAnimation(HDC src1_dc, HDC src2_dc, HDC dst_dc, const char *ef
{
```

```
        unsigned long key = mGEffStr2Key(eff);
        MGEFF_EFFECTOR effector = mGEffEffectorCreate(key);
        MGEFF_SOURCE source1 = mGEffCreateSource(src1_dc);
        MGEFF_SOURCE source2 = mGEffCreateSource(src2_dc);
        MGEFF_SINK sink = mGEffCreateHDCSink(dst_dc);
        MGEFF_ANIMATION handle;

        mGEffEffectorAppendSource(effector, source1);
        mGEffEffectorAppendSource(effector, source2);
        mGEffSetBufferSink(sink, src1_dc);
        mGEffEffectorSetSink(effector, sink);

        handle = mGEffAnimationCreateWithEffector(effector, 0);

        mGEffAnimationSetDuration(handle, duration);

        mGEffAnimationSyncRun(handle);

        mGEffEffectorDelete(effector);

        return 0;
}

static int wndCreate(HWND hwnd, int message, WPARAM wParam, LPARAM lPa
{
        return 0;
}

static int wndPaint(HWND hwnd, HDC hdc, int message, WPARAM wParam, LP
{
        return 0;
}

static int wndKeyDown(HWND hwnd, int message, WPARAM wParam, LPARAM lP
{
        HDC hdc;
        HDC src1, src2;


        hdc = GetClientDC(hwnd);
        src1 = createDCByPicture (hdc, 0xff9abbe3);
        src2 = createDCByPicture (hdc, 0xffd7f1be);

        fillAnimation(src1, src2, hdc, MGEFF_MINOR_leafwindow, 1000);

        DeleteMemDC(src1);
        DeleteMemDC(src2);
```

```
        ReleaseDC(hdc);

        return 0;
}

static int wndProc(HWND hwnd, int message, WPARAM wParam, LPARAM lPara
{
        switch(message)
        {
                case MSG_CREATE:
                mGEffInit();
                wndCreate(hwnd, message, wParam, lParam);
                break;
                case MSG_PAINT:
                {
                        HDC hdc;

                        hdc = BeginPaint(hwnd);
                        wndPaint(hwnd, hdc, message, wParam, lParam);
                        EndPaint(hwnd, hdc);
                }
                break;
                case MSG_KEYDOWN:
                wndKeyDown(hwnd, message, wParam, lParam);
                break;
                case MSG_CLOSE:
                mGEffDeinit();
                DestroyMainWindow (hwnd);
                PostQuitMessage (hwnd);
                break;
                default:
                break;
        }

        return DefaultMainWinProc(hwnd, message, wParam, lParam);
}

int MiniGUIMain(int argc, const char *argv[])
{
        MSG msg;
        HWND hMainHwnd;
        MAINWINCREATE createInfo;

        #ifdef _MGRM_PROCESSES
        JoinLayer(NAME_DEF_LAYER, "animation", 0, 0);
        #endif

        createInfo.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
```

```
        createInfo.dwExStyle = WS_EX_NONE;
        createInfo.spCaption = "effector";
        createInfo.hMenu = 0;
        createInfo.hCursor = GetSystemCursor(0);
        createInfo.hIcon = 0;
        createInfo.MainWindowProc = wndProc;
        createInfo.lx = 0;
        createInfo.ty = 0;
        createInfo.rx = 400;
        createInfo.by = 400;
        createInfo.dwAddData = 0;
        createInfo.hHosting = HWND_DESKTOP;

        hMainHwnd = CreateMainWindow(&createInfo);

        if (hMainHwnd == HWND_INVALID)
        {
                return -1;
        }

        ShowWindow(hMainHwnd, SW_SHOWNORMAL);

        while (GetMessage(&msg, hMainHwnd))
        {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }

        MainWindowThreadCleanup(hMainHwnd);

        return 0;
}
```

从上例可以看到，特效器的使用比较简单，分为创建、指定输入素材源和输出目标、从中创建动画、运行动画、销毁特效器几个步骤。创建特效器及指定输入输出都很简便，这里主要介绍一下后面的步骤：

- 从特效器创建动画 — 通过调用 mGEffAnimationCreateWithEffector 可以从已有的特效器中创建动画对象，返回的动画句柄可能指向单个动画或动画组；
- 运行动画 - 既然从特效器创建出动画了，那么运行动画的方法就与之前介绍过的一样了，可以分同步和异步方式运行，如果是异步动画还可以使用暂停、恢复、等待、停止等 API 控制其生命周期；
- 销毁特效器 — 特效器与动画不同，不会被框架自动回收，必须用户显式调用 mGEffEffectorDelete 进行销毁，否则会造成内存泄漏。

本章小结

本章描述了特效器特性及其使用方法，并介绍了 mGEff 内置特效器的种类和相关属性。通过调用这些内置特效器可以直接产生一些常见特效，但特效器也有不足之处，这里将内置特效器的优缺点

整理如下：

- 优点
  - 使用简单，复用性强
  - 实现上经过优化和测试，不易出错
  - 提供多种版本实现，适用高、中、低不同硬件环境
- 缺点
  - 使用规则单一，大部分特效器只支持从一个素材源到另一个素材源变化的特效实现
  - 特效器输出目标确定后，就只能在固定的目标位置上产生特效，比如做淡入淡出特效时无法只通过特效器实现一边移动一边淡出的效果
  - 可定制性不高，只能实现一些固定的特效

上面的缺点有一部分可以通过自定义特效器来改善，详情请参考“自定义特效器”章节。

---

r6 - 12 Feb 2011 - 16:10:17 - XuBinWang

主窗体动画

主窗体动画

mGEff 支持 MiniGUI 窗口级动画，即以 MiniGUI 窗口为单位进行各种动画变换，通过这个特性用户可以简单快捷的得到窗口级淡出淡入、缩放、滑动等特效。

创建主窗体动画上下文

要使用主窗体动画，需要先创建一个上下文来保存动画相关的信息，其具体原型如下：

```
/** 窗体动画时背景绘制回调原型
* 参数说明:
* hdc - 背景 DC，用户可以往该 DC 上绘制背景
* rc - 背景区域，该区域指定了背景区域 */
typedef void (*SET_BKGND_CB)(HDC hdc, RECT* rc);

/** 窗体前景绘制回调原型
* 参数说明:
* hdc1 - 该 DC 中保存了主窗体的内容截图
* hdc2 - 用户可以往该 DC 中绘制窗口
* rc - 窗口所在的区域信息
*/
typedef void (*DRAW_WND_CB)(HDC hdc1, HDC hdc2, RECT* rc);

/** 创建主窗体动画上下文
* 参数说明:
* duration - 动画的持续时间，单位毫秒
* effector_type - 特效器类型，请参考《特效器》一节了解特效器相关知识，本参数仅对
* curve_type - 曲线类型，请参考《动力曲线》一节了解曲线相关知识，本参数仅对 mGEf
* set_bk - 绘制窗口覆盖的背景区域的回调函数，当调用 mGEffMoveWindow 等函数后窗口
* drawer - 窗口前景绘制回调函数，当通过 mGEffMoveWindow 缩放窗口时，将调用该函数
* 返回值:
* 主窗体动画上下文创建成功将返回其句柄，失败返回 NULL
*/
MGEFF_WINDOW_ANIMATION_CONTEXT
```

```
mGEffCreateWindowAnimationContext(int duration, int effector_type, int
```

创建/显示主窗体

通过主窗体创建动画 API 可以创建并以动画形式显示窗体，其动画时间及动画类型由 mGEffCreateWindowAnimationContext 函数的 duration、effector_type 参数指定：

```
/** 主窗体创建并以动画形式显示
* 参数说明：
* pCreateInfo - 窗体创建信息结构，包含了窗体创建相关的信息
* hctxt - 主窗体动画上下文句柄，包含了产生动画的信息
* 返回值：
* 返回新创建窗体的句柄
*/
HWND mGEffCreateMainWindow(PMAINWINCREATE pCreateInfo, MGEFF_WINDOW_AN

/** 主窗体创建并以动画形式显示
* 参数说明：
* pCreateInfo - 窗体创建信息结构，包含了窗体创建相关的信息
* hctxt - 主窗体动画上下文句柄，包含了产生动画的信息
* cb - 窗口动画创建回调函数，如果用户设置了这个回调，框架将调用该函数来创建及播放动
* 返回值：
* 返回新创建窗体的句柄
*/
HWND mGEffCreateMainWindowEx(PMAINWINCREATE pCreateInfo,
MGEFF_WINDOW_ANIMATION_CONTEXT hctxt, MGEFF_WINDOWANIM_CB cb);

/** 以动画形式显示主窗体
* 参数说明：
* hwnd - 被显示的窗体句柄
* iCmdShow - 窗体显示标志，可以选用的包括 SW_HIDE/SW_SHOW/SW_SHOWNORMAL
* hctxt - 主窗体动画上下文句柄，包含了产生动画的信息
* 返回值：
* 成功返回 TRUE，否则返回 FALSE
*/
BOOL mGEffShowWindow (HWND hwnd, int iCmdShow, MGEFF_WINDOW_ANIMATION_

/** 以动画形式显示主窗体
* 参数说明：
* hwnd - 被显示窗体的句柄
* iCmdShow - 窗体显示标志，可以选用的包括 SW_HIDE/SW_SHOW/SW_SHOWNORMAL
* hctxt - 主窗体动画上下文句柄，包含了产生动画的信息
* cb - 窗口动画创建回调函数，如果用户设置了这个回调，框架将调用该函数来创建及播放动
* 返回值：
* 成功返回 TRUE，否则返回 FALSE
*/
BOOL mGEffShowWindowEx (HWND hwnd, int iCmdShow,
```

```
MGEFF_WINDOW_ANIMATION_CONTEXT hctxt, MGEFF_WINDOWANIM_CB cb);
```

移动主窗体

通过主窗体移动 API 可以将窗体以动画形式进行缩放及移动，其动画时间及移动速度由上下文的 duration、curve_type 指定，移动时背景及前景可以通过 set_bk、drawer 来绘制。

```
/** 以动画形式移动主窗体
* 参数说明：
* hWnd - 被移动窗体的句柄
* x, y, w, h - 将窗体移动到坐标 (x, y) 位置，并将宽、高设为 w, h
* fPaint - 重绘标志，为 TRUE 时将重新绘制窗口
* hctxt - 主窗体动画上下文句柄，包含了产生动画的信息
* 返回值：
* 成功返回 TRUE，否则返回 FALSE
*/
BOOL mGEffMoveWindow (HWND hWnd, int x, int y, int w, int h, BOOL fPai
```

截取主窗体内容

这里介绍一个与主窗体动画相关的函数，可以用来截取窗体内容：

```
/** 获取主窗体内容截图
* 参数说明：
* hwnd - 要截取的窗体句柄
* set_foreground - 窗口前景绘制标记，为 TRUE 时如果窗口是隐藏的则获取截图的同时
* 返回值：
* 成功返回窗口内容截图 DC，失败返回 HDC_INVALID
*/
HDC mGEffGetWindowForeground(HWND hwnd, BOOL set_foreground);
```

使用示例

```
#include <stdio.h>
#include <assert.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/minigui.h>
#include <minigui/control.h>

#include <mgeff/mgeff.h>
```

```
/*****************************************************************
/* main window proc */
static int  mainWindowProc (HWND hWnd, int message, WPARAM wParam, LPA

/*****************************************************************
int MiniGUIMain (int argc, const char *argv[])
{
        HWND hMainHwnd;
        MAINWINCREATE createInfo;
        MSG msg;

        MGEFF_WINDOW_ANIMATION_CONTEXT handle;

        #ifdef _MGRM_PROCESSES
        JoinLayer (NAME_DEF_LAYER, "animation", 0, 0);
        #endif

        createInfo.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
        createInfo.dwExStyle = WS_EX_NONE;
        createInfo.spCaption = "animation_window";
        createInfo.hMenu = 0;
        createInfo.hCursor = GetSystemCursor (0);
        createInfo.hIcon = 0;
        createInfo.MainWindowProc = mainWindowProc;
        createInfo.lx = 0;
        createInfo.ty = 0;
        createInfo.rx = 240;
        createInfo.by = 320;
        createInfo.iBkColor = COLOR_lightwhite;
        createInfo.dwAddData = 0;
        createInfo.hHosting = HWND_DESKTOP;

        handle = mGEffCreateWindowAnimationContext (10000, MGEFF_EFFEC

        hMainHwnd = mGEffCreateMainWindow (&createInfo, handle);

        if (hMainHwnd == HWND_INVALID) {
                return -1;
        }

        mGEffShowWindow (hMainHwnd, SW_SHOWNORMAL, handle);

        while (GetMessage (&msg, hMainHwnd)) {
                TranslateMessage (&msg);
                DispatchMessage (&msg);
        }

        mGEffDestroyWindowAnimationContext (handle);
```

```
        MainWindowThreadCleanup (hMainHwnd);

        return 0;
}

/*************************************************************************
/* main window proc */
static int mainWindowProc (HWND hWnd, int message, WPARAM wParam, LPAR
{
        switch (message) {
                case MSG_CREATE:
                /* init mgeff library */
                mGEffInit ();

                break;

                case MSG_CLOSE:
                DestroyMainWindow (hWnd);
                PostQuitMessage (hWnd);

                /* deinit mgeff library */
                mGEffDeinit ();
                break;

                default:
                break;
        }

        return DefaultMainWinProc (hWnd, message, wParam, lParam);
}
```

本章小结

本章介绍了 mGEff 中的主窗体动画特性，通过这个特性用户可以非常方便的以动画形式显示、移动及缩放 MiniGUI 主窗体，而无须关心 mGEff 和 MiniGUI 的相关接口。

-- ZhaolinHu - 19 Jan 2011

r4 - 27 Jan 2011 - 01:29:48 - XuBinWang

自定义特效器

自定义特效器

如何自定义

实现特效器

注册特效器

特效器上下文

使用示例

```
#include <stdio.h>
#include <assert.h>
#include <string.h>

#include <minigui/common.h>
#include <minigui/gdi.h>
#include <minigui/window.h>
#include <minigui/minigui.h>

#include <mgeff/mgeff.h>

//custom effector define
#define MGEFF_MINOR_custom "custom"
#define MGEFF_EFFECTOR_CUSTOM mGEffStr2Key(MGEFF_MINOR_custom)

typedef struct _EffCustomContext {
        int prev_reach;
} EffCustomContext;
```

```
void eff_get_rect(HDC hdc, RECT* rect)
{
        rect->left = 0;
        rect->top  = 0;
        rect->right  = GetGDCapability(hdc, GDCAP_MAXX) + 1;
        rect->bottom = GetGDCapability(hdc, GDCAP_MAXY) + 1;
}
static MGEFF_EFFECTOR effcustomeffector_init (MGEFF_EFFECTOR effector)
{
        EffCustomContext *context = (EffCustomContext *)calloc (1, siz
        context->prev_reach = 0;
        mGEffEffectorSetContext (effector, context);
        return effector;
}

static void effcustomeffector_finalize (MGEFF_EFFECTOR effector)
{
        EffCustomContext *context = (EffCustomContext *)mGEffEffectorG
        free (context);
}

static void effcustomeffector_ondraw (MGEFF_ANIMATION anim, MGEFF_EFFE
HDC sink_dc, int id, void *value)
{
        EffCustomContext *context = (EffCustomContext *)mGEffEffectorG
        MGEFF_SOURCE src1 = mGEffEffectorGetSource (effector, 0);
        MGEFF_SOURCE src2 = mGEffEffectorGetSource (effector, 1);
        HDC src1_dc = mGEffGetSourceDC (src1);
        HDC src2_dc = mGEffGetSourceDC (src2);
        RECT rc1, rc2, rc_sink;
        eff_get_rect (src1_dc, &rc1);
        eff_get_rect (src2_dc, &rc2);
        eff_get_rect (sink_dc, &rc_sink);

        int reach2 = RECTH(rc2) * (*(float *)value);
        BitBlt (src2_dc, rc2.left, reach2, RECTW(rc2), rc2.bottom,
        sink_dc, 0, 0, 0);
        if (context->prev_reach < reach2) {
                int reach1 = RECTH(rc1) - reach2;
                BitBlt (src1_dc, rc1.left, reach1, RECTW(rc1), rc1.bot
                sink_dc, 0, reach1, 0);
        }

        context->prev_reach = reach2;
}

static void effcustomeffector_begindraw (MGEFF_ANIMATION anim, MGEFF_E
{
```

```
        float s = 1.0;
        float e = 0.0;

        EffCustomContext *context = (EffCustomContext *)mGEffEffectorG
        context->prev_reach = s;
        mGEffAnimationSetStartValue (anim, &s);
        mGEffAnimationSetEndValue (anim, &e);
        mGEffAnimationSetCurve (anim, OutBounce);
}
static void effcustomeffector_enddraw(MGEFF_ANIMATION anim, MGEFF_EFFE
{
}
static int effcustomeffector_setproperty(MGEFF_EFFECTOR effector, int
{
        return 0;
}
static int effcustomeffector_getproperty(MGEFF_EFFECTOR effector, int
{
        return 0;
}

MGEFF_EFFECTOROPS custom = {
        MGEFF_MINOR_custom,
        MGEFF_FLOAT,
        effcustomeffector_init,
        effcustomeffector_finalize,
        effcustomeffector_ondraw,
        effcustomeffector_begindraw,
        effcustomeffector_enddraw,
        effcustomeffector_setproperty,
        effcustomeffector_getproperty,
};
/*-----------------------------------------------------------------
static HDC createDCByPicture(HDC hdc, int color)
{
        HDC dc;
        int w,h;

        w = GetGDCapability(hdc, GDCAP_MAXX) + 1;
        h = GetGDCapability(hdc, GDCAP_MAXY) + 1;
        dc = CreateCompatibleDCEx(hdc, w, h);
        SetBrushColor(dc, color);
        FillBox(dc, 0, 0, w, h);

        return dc;
}

int fillAnimation(HDC src1_dc, HDC src2_dc, HDC dst_dc, const char *ef
```

```
{
        unsigned long key = mGEffStr2Key(eff);
        MGEFF_EFFECTOR effector = mGEffEffectorCreate(key);
        MGEFF_SOURCE source1 = mGEffCreateSource(src1_dc);
        MGEFF_SOURCE source2 = mGEffCreateSource(src2_dc);
        MGEFF_SINK sink = mGEffCreateHDCSink(dst_dc);
        MGEFF_ANIMATION handle;

        mGEffEffectorAppendSource(effector, source1);
        mGEffEffectorAppendSource(effector, source2);
        mGEffSetBufferSink(sink, src1_dc);
        mGEffEffectorSetSink(effector, sink);

        handle = mGEffAnimationCreateWithEffector(effector, 0);

        mGEffAnimationSetDuration(handle, duration);
        mGEffAnimationSyncRun(handle);

        mGEffEffectorDelete(effector);

        return 0;
}

static int wndCreate(HWND hwnd, int message, WPARAM wParam, LPARAM lPa
{
        return 0;
}

static int wndPaint(HWND hwnd, HDC hdc, int message, WPARAM wParam, LP
{
        return 0;
}

static int wndKeyDown(HWND hwnd, int message, WPARAM wParam, LPARAM lP
{
        HDC hdc;
        HDC src1, src2;


        hdc = GetClientDC(hwnd);
        src1 = createDCByPicture (hdc, 0xff9abbe3);
        src2 = createDCByPicture (hdc, 0xffd7f1be);

        fillAnimation(src1, src2, hdc, MGEFF_MINOR_custom, 1000);

        DeleteMemDC(src1);
        DeleteMemDC(src2);
```

```
        ReleaseDC(hdc);

        return 0;
}

static int wndProc(HWND hwnd, int message, WPARAM wParam, LPARAM lPara
{
        switch(message)
        {
                case MSG_CREATE:
                mGEffInit();
                mGEffEffectorRegister(&custom);
                wndCreate(hwnd, message, wParam, lParam);
                break;
                case MSG_PAINT:
                {
                        HDC hdc;

                        hdc = BeginPaint(hwnd);
                        wndPaint(hwnd, hdc, message, wParam, lParam);
                        EndPaint(hwnd, hdc);
                }
                break;
                case MSG_KEYDOWN:
                wndKeyDown(hwnd, message, wParam, lParam);
                break;
                case MSG_CLOSE:
                mGEffEffectorUnRegister(&custom);
                mGEffDeinit();
                DestroyMainWindow (hwnd);
                PostQuitMessage (hwnd);
                break;
                default:
                break;
        }

        return DefaultMainWinProc(hwnd, message, wParam, lParam);
}

int MiniGUIMain(int argc, const char *argv[])
{
        MSG msg;
        HWND hMainHwnd;
        MAINWINCREATE createInfo;

        #ifdef _MGRM_PROCESSES
        JoinLayer(NAME_DEF_LAYER, "animation", 0, 0);
        #endif
```

```
        createInfo.dwStyle = WS_VISIBLE | WS_BORDER | WS_CAPTION;
        createInfo.dwExStyle = WS_EX_NONE;
        createInfo.spCaption = "effector";
        createInfo.hMenu = 0;
        createInfo.hCursor = GetSystemCursor(0);
        createInfo.hIcon = 0;
        createInfo.MainWindowProc = wndProc;
        createInfo.lx = 0;
        createInfo.ty = 0;
        createInfo.rx = 400;
        createInfo.by = 400;
        createInfo.dwAddData = 0;
        createInfo.hHosting = HWND_DESKTOP;

        hMainHwnd = CreateMainWindow(&createInfo);

        if (hMainHwnd == HWND_INVALID)
        {
                return -1;
        }

        ShowWindow(hMainHwnd, SW_SHOWNORMAL);

        while (GetMessage(&msg, hMainHwnd))
        {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }

        MainWindowThreadCleanup(hMainHwnd);

        return 0;
}
```

本章小结

-- KaiDong - 20 Jan 2011

自定义动力曲线

自定义动力曲线

动力曲线本质

如何自定义曲线

定义XX

使用示例

小结

-- KaiDong - 21 Jan 2011

相关数据结构

相关数据结构

- MGEFF_ANIMATION
- EffPoint?

```
typedef struct _EffPoint {
        int x;
        int y;
} EffPoint;
```

- EffPointF?

```
typedef struct _EffPointF {
        float x;
        float y;
} EffPointF;
```

- EffPoint3D?

```
typedef struct _EffPoint3D {
        int x;
        int y;
        int z;
} EffPoint3D;
```

- EffPointF3D?

```
typedef struct _EffPointF3D {
        float x;
        float y;
        float z;
} EffPointF3D;
```

-- XuBinWang - 12 Jan 2011

This topic: Main > MiniGUI > MGEffV1dot0ProgrammingGuide > MGEffV1dot0PGAppendixI
History: r5 - 15 Feb 2011 - 19:09:40 - YongmingWei