Painter (池塘夜降彩色雨)

学院: 光电与计算机学院

专业: 自动化

学生姓名: 许若琪

学号: 1935031028

指导教师: 马立新

完成日期: 2021 年 3 月 18 日

Painter (池塘夜降彩色雨) 题目要求

设计程序演示美丽的"池塘夜降彩色雨"景色: 色彩缤纷的雨点飘飘洒洒的从天而降，滴滴入水有声，溅起圈圈微澜。要求(带*项为选做要求):

1. 雨点的空中出现位置、降落过程的可见程度、入水位置、颜色、最大水圈等，都是随机确定的。
2. 多个雨点按照各自的随机参数和存在状态，同时演示在屏幕上。
3. *增加"电闪雷鸣"景象。
4. *增加风的效果，展现"风雨飘摇"的情景。
5. *增加雨点密度的变化: 时而"和风细雨"，时而"暴风骤雨"。
6. *将"池塘"改为"荷塘"，雨点落在荷叶上的效果是溅起四散的水珠，响声也不同。

内容列表

题目分析

- 综合看题目要求，本题须使用C语言绘制图像，考虑到C语言可用的主要图形库有EasyX和OpenGL，其中OpenGL主要用来编辑3维图像，本题中并无此需求，故选用EasyX进行开发。
- 在目标全要求(基本要求与选做要求宣布实现)的基础上看题目，题目所要求展示的各个视觉元素可进行模块化分离(背景、荷叶、雨滴、涟漪、水花、闪电)，可分别独立开发，最后整合。

概要设计及细节构思

- 代码严格使用下划线命名法。
- 常用表达式和常量用宏定义代替。
- 应当分离题目的函数与入口(main函数)，题目使用到的函数应当在main函数所在的文件之外进行声明和定义。
- EasyX仅支持对图像的基本操作(如基本几何图形的绘制，像素点绘制，获取某坐标下像素的RGB值...)，绘制题目中所需的视觉元素需要多个几何图形的组合和刷新，则可对不同视觉元素分别进行编写和封装。
- 根据不同视觉元素所需的基本参数分别设计结构体。
- 每个视觉元素均包含四个状态：创建句柄、初始化、绘制与刷新、结束。可依照这四个阶段来设计用于视觉元素的各个函数。
- 使用雨滴下落速度表示不同距离的雨滴，则越远的雨滴下落速度越慢，结束高度越高，涟漪大小越小，水花大小越小。
- 雨滴、涟漪、水花这样的视觉元素总是同时出现多个，且数量与动画状态均不定，实现这样的功能有两种思路：一种是在每一次要产生雨滴/涟漪/水花时都为其申请动态内存(malloc)，待他们动画结束之后再回收内存(free)；一种是一开始就申请数组存放所有的结构体，然后在某一时刻选择调用它们中的某些成员。考虑在程序运转过程中会产生大量的雨滴/涟漪/水花，频繁的申请与释放动态内存会造成内存碎片大量堆积，故选用第二种方法。
- 对于雨滴/涟漪/水花这样的句柄为数组的视觉元素，因为初始化的方法较复杂，应当使用工厂函数来初始化句柄并返回。
- 每个函数都应通过依赖注入(非单例模式，无全局变量)的形式工作，以减少程序耦合度。

- 应当设置config.h头文件用来调整库的部分表现效果(画面的长度与宽度，最大雨滴数，初始雨滴数.....)
- 应当可以选择性的绘制某些视觉元素，并且使用统一的函数来调用这些功能。
- 考虑函数的可见范围，对于用户使用不到的函数应该使其静态化，并且不在头文件内暴露其接口。

详细设计

1. HSL颜色信息结构体

```
typedef struct {
    float H;
    float S;
    float L;
} HSL_Color_Type;
```

2. 雨滴结构体及其相关函数

```
typedef struct {
    char label[20];  // 结构体标签/名称
    int position_start[2];    // 起始坐标 [x, y]
    int position_height_end;  // 终止高度 [x, y]
    int step;                // 雨滴下落速度
    int is_on : 1;             // 是否落下(被画出)
    HSL_Color_Type color_HSL;  // 颜色
} Raindrop_Type;

/**
 * @brief 雨滴初始化函数(静态函数)
 *
 * @param raindrop  雨滴句柄
 * @param x 雨滴出生位置 X轴坐标
 * @param y 雨滴出生位置 Y轴坐标
 * @param height_end 雨滴结束下落时的 Y轴坐标
 * @param step 雨滴下落速度
 * @param H 雨滴HSL H
 * @param S 雨滴HSL S
 * @param L 雨滴HSL L
 * @param is_on 该雨滴是否要被绘制
 */
static Raindrop_Type* Raindrop_Init(Raindrop_Type* raindrop, int x, int y,
                                    int height_end, int step, float H, float S,
```

```
                                    float L, int is_on);
```

3. 荷叶结构体及其相关函数

```
typedef struct {
    char label[20];  // 结构体标签/名称
    int position_rect[4];      // 椭圆坐标 [left, top, right, bottom]
    HSL_Color_Type color_HSL;  // 颜色
} Lotus_Leaf_Type;

/**
 * @brief 荷叶初始化函数
 *
 * @param lotusLeaf  荷叶句柄
 * @param left 荷叶绘制位置 左上角X轴坐标
 * @param top 荷叶绘制位置 左上角Y轴坐标
 * @param right 荷叶绘制位置 右下角X轴坐标
 * @param bottom 荷叶绘制位置 右下角Y轴坐标
 * @param H 荷叶HSL H
 * @param S 荷叶HSL S
 * @param L 荷叶HSL L
 */
void Lotus_Leaf_Init(Lotus_Leaf_Type* lotusLeaf, int left, int top,
                     int right, int bottom, float H, float S,
                     float L);
/**
 * @brief 荷叶绘制函数(静态函数)
 *
 * @param lotusLeaf  荷叶句柄
*/
static void Lotus_Leaf_Draw(Lotus_Leaf_Type* lotusLeaf);
```

4. 风结构体及其相关函数

```
typedef struct {
    char label[20];  // 结构体标签/名称
    int is_on : 1;  // 是否有风
    int speed;      // 风速
} Wind_Type;

/**
 * @brief 风初始化函数
 *
 * @param wind  风句柄
```

```
 * @param is_on  是否开启风
 * @param speed  风速
 */
void Wind_Init(Wind_Type* wind, int is_on, int speed);

/**
 * @brief 切换风开启状态(静态函数)
 *
 * @param wind  风句柄
 * @param is_on  是否开启风
 */
static void Wind_Switch(Wind_Type* wind, int on_off);
```

5. 涟漪结构体及其相关函数

```
typedef struct {
    char label[20];  // 结构体标签/名称
    int raindrop_speed; // 雨滴速度
    int is_on : 1; // 是否绘制此涟漪
    int position_rect[4];  // 椭圆坐标 [left, top, right, bottom]
} Ripple_Type;

/**
 * @brief 涟漪初始化函数(静态函数)
 *
 * @param ripple 荷叶句柄
 * @param left 涟漪绘制位置 左上角X轴坐标
 * @param top 涟漪绘制位置 左上角Y轴坐标
 * @param right 涟漪绘制位置 右下角X轴坐标
 * @param bottom 涟漪绘制位置 右下角Y轴坐标
 * @param raindrop_speed 雨滴下落速度
 * @param is_on 是否绘制此涟漪
 */
static void Ripple_Init(Ripple_Type* ripple, int left, int top, int right,
                        int bottom, int raindrop_speed, int is_on);

/**
 * @brief 涟漪工厂函数
 */
Ripple_Type* Ripples_Create();

/**
 * @brief 涟漪注册启动函数(静态函数)
 *
```

```
 * @param ripple 涟漪句柄
 * @param left 涟漪绘制位置 左上角X轴坐标
 * @param top 涟漪绘制位置 左上角Y轴坐标
 * @param right 涟漪绘制位置 右下角X轴坐标
 * @param bottom 涟漪绘制位置 右下角Y轴坐标
 * @param raindrop_speed 雨滴下落速度
 */
static void Ripples_Register(Ripple_Type* ripples, int left, int top, int right,
                             int bottom, int raindrop_speed);

/**
 * @brief 涟漪绘制函数(静态函数)
 *
 * @param ripple 涟漪句柄
 */
static void Ripples_Draw(Ripple_Type* ripples);
```

6. 水花结构体及其相关函数

```
typedef struct {
    char label[20];  // 结构体标签/名称
    int raindrop_speed; // 雨滴速度
    int is_on : 1; // 是否绘制此水花
    int positioin_start[2];  // 起始坐标 [x, y]
} Spray_Type;

/**
 * @brief 水花初始化函数(静态函数)
 *
 * @param spray 水花句柄
 * @param x 水花出生位置 X轴坐标
 * @param y 水花出生位置 Y轴坐标
 * @param raindrop_speed 雨滴下落速度
 * @param is_on 是否绘制此水花
 */
static void Spray_Init(Spray_Type* spray, int x, int y, int raindrop_speed,
                       int is_on);

/**
 * @brief 水花工厂函数
 */
Spray_Type* Sprays_Create();

/**
```

```
 * @brief 水花注册启动函数(静态函数)
 *
 * @param spray 水花句柄
 * @param x 水花出生位置 X轴坐标
 * @param y 水花出生位置 Y轴坐标
 * @param raindrop_speed 雨滴下落速度
 */
static void Sprays_Register(Spray_Type* sprays, int x, int y,
                            int raindrop_speed);
/**
 * @brief 水花绘制函数(静态函数)
 *
 * @param spray 水花句柄
 */
static void Sprays_Draw(Spray_Type* sprays);
```

7. 雨相关函数

```
/**
 * @brief 雨数组(句柄)工厂函数
 */
Raindrop_Type* Rain_Create();

/**
 * @brief 雨绘制函数(静态函数)
 *
 * @param rain 雨句柄
 * @param rqrd_raindrop_num  所绘制雨滴的数量
 * @param ripples 涟漪句柄
 * @param sprays 水花句柄
 * @param wind 风句柄
 */
static int Rain_Draw(Raindrop_Type* rain, int rqrd_raindrop_num,
                     Ripple_Type* , Spray_Type* ,
                     Wind_Type* wind)
```

8. Painter相关函数

```
/**
 * @brief Painter初始化函数
 *
 * @param width 画布宽度
 * @param height 画布高度
 */
```

```c
void Painter_Init(int width, int height);

/**
 * @brief Painter指令台
 *
 * @param handle 主要操作句柄
 * @param wind 风句柄
 * @param ripples 涟漪句柄
 * @param sprays 水花句柄
 * @param cmd 指令字符串
 */
void Painter_CMD(void* handle, Wind_Type* wind, Ripple_Type* ripples,
                 Spray_Type* sprays, const char* cmd);

/**
 * @brief Painter画面更新函数
 *
 * @param delay_ms 画面间刷新时间(单位毫秒)
 */
void Painter_Update(int delay_ms);

/**
 * @brief Painter结束释放函数
 *
 * @param rain 雨句柄
 * @param ripples 涟漪句柄
 * @param sprays 水花句柄
 */
void Painter_End(Raindrop_Type* rain, Ripple_Type* ripples, Spray_Type* sprays);
```

9. 常用宏定义

```c
#define TRUE 1
#define FALSE 0

#define CMD_IS_SAME(cmd_a, cmd_b) (strcmp((cmd_a), (cmd_b)) == 0) // 指令选择

#define MAX(a, b) (((a) > (b)) ? (a) : (b)) // 返回大
#define MIN(a, b) (((a) < (b)) ? (a) : (b)) // 返回小
```

具体代码

1. 常用宏定义: marco.h

```
#ifndef __MARCO_H
#define __MARCO_H

#include <string.h>

#define TRUE 1
#define FALSE 0

#define CMD_IS_SAME(cmd_a, cmd_b) (strcmp((cmd_a), (cmd_b)) == 0)

#define MAX(a, b) (((a) > (b)) ? (a) : (b))
#define MIN(a, b) (((a) < (b)) ? (a) : (b))

#endif
```

2. Painter头文件: painter.h

```
#ifndef __PAINTER_H
#define __PAINTER_H

#include "config.h"
#include "marco.h"

// HSL颜色结构体
typedef struct {
    float H;
    float S;
    float L;
} HSL_Color_Type;

// 雨滴结构体
typedef struct {
    char label[20];  // 结构体标签/名称

    int position_start[2];    // 起始坐标 [x, y]
    int position_height_end;  // 终止高度 [x, y]

    int step;
```

```
    int is_on : 1;              // 是否落下(被画出)
    HSL_Color_Type color_HSL;  // 颜色
} Raindrop_Type;

Raindrop_Type* Rain_Create();

// 荷叶结构体
typedef struct {
    char label[20];  // 结构体标签/名称

    int position_rect[4];      // 椭圆坐标 [left, top, right, bottom]
    HSL_Color_Type color_HSL;  // 颜色
} Lotus_Leaf_Type;

// 风结构体
typedef struct {
    char label[20];  // 结构体标签/名称

    int is_on : 1;  // 是否有风
    int speed;      // 速度
} Wind_Type;

// 涟漪结构体
typedef struct {
    char label[20];  // 结构体标签/名称

    int raindrop_speed;
    int is_on : 1;
    int position_rect[4];  // 椭圆坐标 [left, top, right, bottom]

} Ripple_Type;

// 水花结构体
typedef struct {
    char label[20];  // 结构体标签/名称

    int raindrop_speed;
    int is_on : 1;
    int positioin_start[2];  // 起始坐标 [x, y]

} Spray_Type;

/**
 * @brief 荷叶初始化函数
 *
```

```
 * @param lotusLeaf  荷叶句柄
 * @param left 荷叶绘制位置 左上角X轴坐标
 * @param top 荷叶绘制位置 左上角Y轴坐标
 * @param right 荷叶绘制位置 右下角X轴坐标
 * @param bottom 荷叶绘制位置 右下角Y轴坐标
 * @param H 荷叶HSL H
 * @param S 荷叶HSL S
 * @param L 荷叶HSL L
 */
void Lotus_Leaf_Init(Lotus_Leaf_Type*, int left, int top, int right, int bottom,
                     float H, float S, float L);

/**
 * @brief 风初始化函数
 *
 * @param wind  风句柄
 * @param is_on  是否开启风
 * @param speed  风速
 */
void Wind_Init(Wind_Type*, int is_on, int speed);

/**
 * @brief 涟漪工厂函数
 */
Ripple_Type* Ripples_Create();

/**
 * @brief 水花工厂函数
 */
Spray_Type* Sprays_Create();

/**
 * @brief Painter初始化函数
 *
 * @param width 画布宽度
 * @param height 画布高度
 */
void Painter_Init(int width, int height);

/**
 * @brief Painter指令台
 *
 * @param handle 主要操作句柄
 * @param wind 风句柄
 * @param ripples 涟漪句柄
```

```
 * @param sprays 水花句柄
 * @param cmd 指令字符串
 */
void Painter_CMD(void* handle, Wind_Type* wind, Ripple_Type* ripples,
                 Spray_Type* sprays, const char* cmd);

/**
 * @brief Painter画面更新函数
 *
 * @param delay_ms 画面间刷新时间(单位毫秒)
 */
void Painter_Update(int delay_ms);

/**
 * @brief Painter结束释放函数
 *
 * @param rain 雨句柄
 * @param ripples 涟漪句柄
 * @param sprays 水花句柄
 *
void Painter_End(Raindrop_Type* rain, Ripple_Type* ripples, Spray_Type* sprays);
#endif
```

3. Painter源文件: painter.cpp

```
#include "painter.h"

#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#pragma comment(lib, "Winmm.lib")

/**
 * @brief 雨滴初始化函数(静态函数)
 *
 * @param raindrop  雨滴句柄
 * @param x 雨滴出生位置 X轴坐标
 * @param y 雨滴出生位置 Y轴坐标
 * @param height_end 雨滴结束下落时的 Y轴坐标
 * @param step 雨滴下落速度
```

```
 * @param H 雨滴颜色 H
 * @param S 雨滴颜色 S
 * @param L 雨滴颜色 L
 * @param is_on 该雨滴是否要被绘制
 */
static Raindrop_Type* Raindrop_Init(Raindrop_Type* raindrop, int x, int y,
                                    int height_end, int step, float H, float S,
                                    float L, int is_on) {
    strcpy_s(raindrop->label, 9, "Raindrop");

    raindrop->position_start[0] = x;
    raindrop->position_start[1] = y;

    raindrop->position_height_end = height_end;

    raindrop->step = step;

    raindrop->color_HSL.H = H;
    raindrop->color_HSL.S = S;
    raindrop->color_HSL.L = L;

    raindrop->is_on = is_on;

    return raindrop;
}

/**
 * @brief 雨数组(句柄)工厂函数
 */
Raindrop_Type* Rain_Create() {
    Raindrop_Type* rain =
        (Raindrop_Type*)malloc(MAX_RAINDROP_NUM * sizeof(Raindrop_Type));
    for (int i = 0; i < MAX_RAINDROP_NUM; i++) {
        int x = rand() % (5 * WIDTH) - WIDTH;
        int y = rand() % (HEIGHT / 3);
        int step = (rand() % 5000) / 1000.0 + 1;
        int height_end = rand() % (HEIGHT / 3) + ((HEIGHT / 3) * 2) * step / 5;
        int H = rand() % 360;
        float S = (float)(rand() % 90) + 0.1;
        float L = step + 3;

        int is_on = TRUE;
        if (i >= START_RAINDROP_NUM) {
            is_on = FALSE;
        }
```

```
        Raindrop_Init(rain + i, x, y, height_end, step, H, S, L, is_on);
    }
    return rain;
}

/**
 * @brief 荷叶绘制函数(静态函数)
 *
 * @param lotusLeaf  荷叶句柄
 */
static void Lotus_Leaf_Draw(Lotus_Leaf_Type* lotusLeaf) {
    setlinecolor(HSLtoRGB(lotusLeaf->color_HSL.H, lotusLeaf->color_HSL.S,
                          MAX(lotusLeaf->color_HSL.L - 0.2f, 0)));
    setfillstyle(BS_SOLID);
    setfillcolor(HSLtoRGB(lotusLeaf->color_HSL.H, lotusLeaf->color_HSL.S,
                          lotusLeaf->color_HSL.L));
    fillellipse(lotusLeaf->position_rect[0], lotusLeaf->position_rect[1],
                lotusLeaf->position_rect[2], lotusLeaf->position_rect[3]);
}

/**
 * @brief 风初始化函数
 *
 * @param wind  风句柄
 * @param is_on  是否开启风
 * @param speed  风速
 */
void Wind_Init(Wind_Type* wind, int is_on, int speed) {
    strcpy_s(wind->label, 5, "Wind");

    wind->is_on = is_on;
    wind->speed = speed;
}

/**
 * @brief 切换风开启状态(静态函数)
 *
 * @param wind  风句柄
 * @param is_on  是否开启风
 */
static void Wind_Switch(Wind_Type* wind, int on_off) { wind->is_on = on_off; }

/**
 * @brief 涟漪初始化函数(静态函数)
```

```
 *
 * @param ripple 荷叶句柄
 * @param left 涟漪绘制位置 左上角X轴坐标
 * @param top 涟漪绘制位置 左上角Y轴坐标
 * @param right 涟漪绘制位置 右下角X轴坐标
 * @param bottom 涟漪绘制位置 右下角Y轴坐标
 * @param raindrop_speed 雨滴下落速度
 * @param is_on 是否绘制此涟漪
 */
static void Ripple_Init(Ripple_Type* ripple, int left, int top, int right,
                        int bottom, int raindrop_speed, int is_on) {
    strcpy_s(ripple->label, 7, "Ripple");

    ripple->position_rect[0] = left;
    ripple->position_rect[1] = top;
    ripple->position_rect[2] = right;
    ripple->position_rect[3] = bottom;

    ripple->raindrop_speed = raindrop_speed;

    ripple->is_on = is_on;
}

/**
 * @brief 涟漪工厂函数
 */
Ripple_Type* Ripples_Create() {
    Ripple_Type* ripples =
        (Ripple_Type*)malloc(MAX_RAINDROP_NUM * sizeof(Ripple_Type));
    for (int i = 0; i < MAX_RAINDROP_NUM; i++) {
        Ripple_Init(ripples + i, 0, 0, 0, 0, 0, FALSE);
    }

    return ripples;
}

/**
 * @brief 涟漪注册启动函数(静态函数)
 *
 * @param ripple 涟漪句柄
 * @param left 涟漪绘制位置 左上角X轴坐标
 * @param top 涟漪绘制位置 左上角Y轴坐标
 * @param right 涟漪绘制位置 右下角X轴坐标
 * @param bottom 涟漪绘制位置 右下角Y轴坐标
 * @param raindrop_speed 雨滴下落速度
```

```
 */
static void Ripples_Register(Ripple_Type* ripples, int left, int top, int right,
                             int bottom, int raindrop_speed) {
    for (int i = 0; i < MAX_RAINDROP_NUM; i++) {
        if (!(ripples + i)->is_on) {
            Ripple_Init(ripples + i, left, top, right, bottom, raindrop_speed,
                        TRUE);
            break;
        }
    }
}

/**
 * @brief 涟漪绘制函数(静态函数)
 *
 * @param ripple 涟漪句柄
 */
static void Ripples_Draw(Ripple_Type* ripples) {
    for (int i = 0; i < MAX_RAINDROP_NUM; i++) {
        if ((ripples + i)->is_on) {
            setlinecolor(0);
            ellipse((ripples + i)->position_rect[0],
                    (ripples + i)->position_rect[1],
                    (ripples + i)->position_rect[2],
                    (ripples + i)->position_rect[3]);
        }

        if ((ripples + i)->is_on) {
            setlinecolor(RGB((ripples + i)->raindrop_speed * 20,
                             (ripples + i)->raindrop_speed * 20,
                             (ripples + i)->raindrop_speed * 20));

            (ripples + i)->position_rect[0] -=
                2 * (ripples + i)->raindrop_speed;
            (ripples + i)->position_rect[1] -= (ripples + i)->raindrop_speed;
            (ripples + i)->position_rect[2] +=
                2 * (ripples + i)->raindrop_speed;
            (ripples + i)->position_rect[3] += (ripples + i)->raindrop_speed;

            ellipse((ripples + i)->position_rect[0],
                    (ripples + i)->position_rect[1],
                    (ripples + i)->position_rect[2],
                    (ripples + i)->position_rect[3]);

            (ripples + i)->raindrop_speed -= 1;
```

```
            if ((ripples + i)->raindrop_speed <= 0) {
                (ripples + i)->is_on = FALSE;
                setlinecolor(0);
                ellipse((ripples + i)->position_rect[0],
                        (ripples + i)->position_rect[1],
                        (ripples + i)->position_rect[2],
                        (ripples + i)->position_rect[3]);
            }
        }
    }
}

/**
 * @brief 水花初始化函数(静态函数)
 *
 * @param spray 水花句柄
 * @param x 水花出生位置 X轴坐标
 * @param y 水花出生位置 Y轴坐标
 * @param raindrop_speed 雨滴下落速度
 * @param is_on 是否绘制此水花
 */
static void Spray_Init(Spray_Type* spray, int x, int y, int raindrop_speed,
                       int is_on) {
    strcpy_s(spray->label, 6, "Spray");

    spray->positioin_start[0] = x;
    spray->positioin_start[1] = y;

    spray->raindrop_speed = raindrop_speed;

    spray->is_on = is_on;
}

/**
 * @brief 水花工厂函数
 */
Spray_Type* Sprays_Create() {
    Spray_Type* sprays =
        (Spray_Type*)malloc(MAX_RAINDROP_NUM * sizeof(Spray_Type));
    for (int i = 0; i < MAX_RAINDROP_NUM; i++) {
        Spray_Init(sprays + i, 0, 0, 0, FALSE);
    }

    return sprays;
```

```
}

/**
 * @brief 水花注册启动函数(静态函数)
 *
 * @param spray 水花句柄
 * @param x 水花出生位置 X轴坐标
 * @param y 水花出生位置 Y轴坐标
 * @param raindrop_speed 雨滴下落速度
 */
static void Sprays_Register(Spray_Type* sprays, int x, int y,
                            int raindrop_speed) {
    for (int i = 0; i < MAX_RAINDROP_NUM; i++) {
        if (!(sprays + i)->is_on) {
            Spray_Init(sprays + i, x, y, raindrop_speed, TRUE);
            break;
        }
    }
}

/**
 * @brief 水花绘制函数(静态函数)
 *
 * @param spray 水花句柄
 */
static void Sprays_Draw(Spray_Type* sprays) {
    for (int i = 0; i < MAX_RAINDROP_NUM; i++) {
        int step = MAX(10 - (sprays + i)->raindrop_speed, 0);

        if ((sprays + i)->is_on) {
            putpixel((sprays + i)->positioin_start[0] - step,
                     (sprays + i)->positioin_start[1] - step, 0);
            putpixel((sprays + i)->positioin_start[0] + step,
                     (sprays + i)->positioin_start[1] - step, 0);
            putpixel((sprays + i)->positioin_start[0],
                     (sprays + i)->positioin_start[1] - step, 0);
        }

        if ((sprays + i)->is_on) {
            (sprays + i)->raindrop_speed -= 1;
            step = MAX(10 - (sprays + i)->raindrop_speed, 0);

            putpixel((sprays + i)->positioin_start[0] - step,
                     (sprays + i)->positioin_start[1] - step,
                     RGB(step * 20, step * 20, step * 20));
```

```
            putpixel((sprays + i)->positioin_start[0] + step,
                     (sprays + i)->positioin_start[1] - step,
                     RGB(step * 20, step * 20, step * 20));
            putpixel((sprays + i)->positioin_start[0],
                     (sprays + i)->positioin_start[1] - step,
                     RGB(step * 20, step * 20, step * 20));

            if ((sprays + i)->raindrop_speed <= 0) {
                (sprays + i)->is_on = FALSE;
                putpixel((sprays + i)->positioin_start[0] - step,
                         (sprays + i)->positioin_start[1] - step, 0);
                putpixel((sprays + i)->positioin_start[0] + step,
                         (sprays + i)->positioin_start[1] - step, 0);
                putpixel((sprays + i)->positioin_start[0],
                         (sprays + i)->positioin_start[1] - step, 0);
            }
        }
    }
}

/**
 * @brief Painter初始化函数
 *
 * @param width 画布宽度
 * @param height 画布高度
 */
void Painter_Init(int width, int height) {
    srand((unsigned)time(NULL));  // 随机种子
    initgraph(width, height);     // 创建绘图窗口
}

/**
 * @brief 雨绘制函数(静态函数)
 *
 * @param rain 雨句柄
 * @param rqrd_raindrop_num  所绘制雨滴的数量
 * @param ripples 涟漪句柄
 * @param sprays 水花句柄
 * @param wind 风句柄
 */
static int Rain_Draw(Raindrop_Type* rain, int rqrd_raindrop_num,
                     Ripple_Type* ripples, Spray_Type* sprays,
                     Wind_Type* wind) {
    static int is_first_raindrop = TRUE;
    static int real_raindrop_num = START_RAINDROP_NUM;
```

```
    static int raindrop_leaf_count = 0;

    for (int i = 0; i < MAX_RAINDROP_NUM; i++) {
        putpixel((rain + i)->position_start[0], (rain + i)->position_start[1],
                 0);
        if ((rain + i)->is_on) {
            (rain + i)->position_start[1] =
                MIN((rain + i)->position_start[1] + (rain + i)->step,
                    (rain + i)->position_height_end);

            if (wind->is_on) {
                (rain + i)->position_start[0] += (rain + i)->step;
            }
        }

        if ((rain + i)->is_on &&
            (rain + i)->position_start[1] >= (rain + i)->position_height_end) {
            (rain + i)->is_on = FALSE;
            real_raindrop_num--;
            //*************************** 雨滴重生 ***************************
            int x = rand() % WIDTH;
            int y = rand() % (HEIGHT / 3);
            int height_end = rand() % (HEIGHT / 3) + ((HEIGHT / 3) * 2);
            int step = (rand() % 5000) / 1000.0 + 1;
            int H = rand() % 360;
            float S = (float)(rand() % 90) + 0.1;
            float L = step + 3;

            int is_on = TRUE;
            if (real_raindrop_num >= rqrd_raindrop_num) {
                is_on = FALSE;
            }
            real_raindrop_num += is_on;

            if (getpixel((rain + i)->position_start[0],
                         (rain + i)->position_start[1]) != 0) {
                if (++raindrop_leaf_count > 25) {
                    mciSendString("close rl", NULL, 0, NULL);
                    mciSendString("open raindrop_leaf.mp3 alias rl", NULL, 0,
                                  NULL);
                    mciSendString("play rl", NULL, 0, NULL);
                    raindrop_leaf_count = 0;
                }

                Sprays_Register(sprays, (rain + i)->position_start[0],
```

```
                                (rain + i)->position_start[1],
                                (rain + i)->step);
            } else {
                if (is_first_raindrop) {
                    mciSendString("open raindrop_water.mp3 alias rw", NULL, 0,
                                  NULL);
                    mciSendString("play rw repeat", NULL, 0, NULL);
                    is_first_raindrop = FALSE;
                }

                Ripples_Register(ripples, (rain + i)->position_start[0],
                                 (rain + i)->position_start[1],
                                 (rain + i)->position_start[0],
                                 (rain + i)->position_start[1],
                                 (rain + i)->step);
            }

            Raindrop_Init((rain + i), x, y, height_end, step, H, S, L, is_on);
        }
        if ((!(rain + i)->is_on) && real_raindrop_num < rqrd_raindrop_num) {
            (rain + i)->is_on = TRUE;
            real_raindrop_num++;
        }

        if ((rain + i)->is_on) {
            putpixel((rain + i)->position_start[0],
                     (rain + i)->position_start[1],
                     HSLtoRGB((rain + i)->color_HSL.H, (rain + i)->color_HSL.S,
                              (rain + i)->color_HSL.L));
        }
    }

    return real_raindrop_num;
}

/**
 * @brief 荷叶初始化函数
 *
 * @param lotusLeaf  荷叶句柄
 * @param left 荷叶绘制位置 左上角X轴坐标
 * @param top 荷叶绘制位置 左上角Y轴坐标
 * @param right 荷叶绘制位置 右下角X轴坐标
 * @param bottom 荷叶绘制位置 右下角Y轴坐标
 * @param H 荷叶HSL H
 * @param S 荷叶HSL S
```

```
 * @param L 荷叶HSL L
 */
void Lotus_Leaf_Init(Lotus_Leaf_Type* lotusLeaf, int left, int top, int right,
                     int bottom, float H, float S, float L) {
    strcpy_s(lotusLeaf->label, 11, "Lotus_Leaf");

    lotusLeaf->position_rect[0] = left;
    lotusLeaf->position_rect[1] = top;
    lotusLeaf->position_rect[2] = right;
    lotusLeaf->position_rect[3] = bottom;

    lotusLeaf->color_HSL.H = H;
    lotusLeaf->color_HSL.S = S;
    lotusLeaf->color_HSL.L = L;
}

static void Flash_Bang_Draw() {
    setfillcolor(WHITE);
    solidrectangle(0, 0, WIDTH, HEIGHT);

    mciSendString("close fb", NULL, 0, NULL);
    mciSendString("open flash_bang.mp3 alias fb", NULL, 0, NULL);
    mciSendString("play fb", NULL, 0, NULL);

    Sleep(50);
    setfillcolor(BLACK);
    solidrectangle(0, 0, WIDTH, HEIGHT);
    Sleep(20);
    setfillcolor(WHITE);
    solidrectangle(0, 0, WIDTH, HEIGHT);
    setfillcolor(BLACK);
    solidrectangle(0, 0, WIDTH, HEIGHT);
}

/**
 * @brief Painter指令台
 *
 * @param handle 主要操作句柄
 * @param wind 风句柄
 * @param ripples 涟漪句柄
 * @param sprays 水花句柄
 * @param cmd 指令字符串
 */
void Painter_CMD(void* handle, Wind_Type* wind, Ripple_Type* ripples,
                 Spray_Type* sprays, const char* cmd) {
```

```
    static int rqrd_raindrop_num = 40;

    if (CMD_IS_SAME(cmd, "lotus_leaf")) {
        Lotus_Leaf_Draw((Lotus_Leaf_Type*)handle);
    } else if (CMD_IS_SAME(cmd, "rain")) {
        Rain_Draw((Raindrop_Type*)handle, rqrd_raindrop_num, ripples, sprays,
                  wind);
    } else if (CMD_IS_SAME(cmd, "wind_on")) {
        Wind_Switch(wind, TRUE);
    } else if (CMD_IS_SAME(cmd, "wind_off")) {
        Wind_Switch(wind, FALSE);
    } else if (CMD_IS_SAME(cmd, "ripples")) {
        Ripples_Draw(ripples);
    } else if (CMD_IS_SAME(cmd, "sprays")) {
        Sprays_Draw(sprays);
    } else if (CMD_IS_SAME(cmd, "raindrop_more")) {
        rqrd_raindrop_num += rand() % 35 + 45;
        rqrd_raindrop_num = MIN(rqrd_raindrop_num, MAX_RAINDROP_NUM);
    } else if (CMD_IS_SAME(cmd, "raindrop_less")) {
        rqrd_raindrop_num -= rand() % 45 + 35;
        rqrd_raindrop_num = MAX(rqrd_raindrop_num, 20);
    } else if (CMD_IS_SAME(cmd, "flash_bang")) {
        Flash_Bang_Draw();
    }
}

/**
 * @brief Painter画面更新函数
 *
 * @param delay_ms 画面间刷新时间(单位毫秒)
 */
void Painter_Update(int delay_ms) { Sleep(delay_ms); }

/**
 * @brief Painter结束释放函数
 *
 * @param rain 雨句柄
 * @param ripples 涟漪句柄
 * @param sprays 水花句柄
 */
void Painter_End(Raindrop_Type* rain, Ripple_Type* ripples,
                 Spray_Type* sprays) {
    free(rain);
    free(ripples);
    free(sprays);
```

```
    closegraph();
}
```

4. painter参数文件: config.h

```
#ifndef __CONFIG_H
#define __CONFIG_H

#define MAX_RAINDROP_NUM 120
#define START_RAINDROP_NUM (MAX_RAINDROP_NUM / 2)

#define WIDTH 300
#define HEIGHT 225

#endif
```

5. 入口函数所在文件: main.cpp

```
#include <conio.h>
#include <stdlib.h>

#include "painter.h"

int main() {
    register int timer = 0;
    register int wind_flag = TRUE;
    Painter_Init(WIDTH, HEIGHT);

    Wind_Type wind;

    Lotus_Leaf_Type Leaf_A;
    Lotus_Leaf_Type Leaf_B;
    Lotus_Leaf_Type Leaf_C;
    Lotus_Leaf_Type Leaf_D;
    Lotus_Leaf_Type Leaf_E;
    Lotus_Leaf_Type Leaf_F;

    Raindrop_Type* Rain = Rain_Create();
    Ripple_Type* Ripples = Ripples_Create();
    Spray_Type* Sprays = Sprays_Create();

    Lotus_Leaf_Init(&Leaf_A, 40, 120, 100, 150, 92, 0.5f, 0.5f);
    Lotus_Leaf_Init(&Leaf_B, 80, 140, 140, 170, 92, 0.5f, 0.5f);
    Lotus_Leaf_Init(&Leaf_C, 30, 165, 90, 195, 92, 0.5f, 0.5f);
```

```
    Lotus_Leaf_Init(&Leaf_D, 130, 185, 190, 215, 92, 0.5f, 0.5f);
    Lotus_Leaf_Init(&Leaf_E, 200, 140, 260, 170, 92, 0.5f, 0.5f);
    Lotus_Leaf_Init(&Leaf_F, 240, 175, 300, 205, 92, 0.5f, 0.5f);

    Wind_Init(&wind, FALSE, 1);

    // 按任意键退出
    while (!_kbhit()) {
        if ((timer++) > (500 + (rand() % 300))) {
            if (wind_flag) {
                Painter_CMD(NULL, &wind, NULL, NULL, "wind_off");
                Painter_CMD(NULL, &wind, NULL, NULL, "raindrop_less");
            } else {
                Painter_CMD(NULL, &wind, NULL, NULL, "wind_on");
                Painter_CMD(NULL, &wind, NULL, NULL, "raindrop_more");
            }
            wind_flag = !wind_flag;
            timer = 0;
        }

        if ((!(timer % 250)) && (rand() % 2)) {
            Painter_CMD(NULL, NULL, NULL, NULL, "flash_bang");
        }

        Painter_CMD(NULL, NULL, Ripples, NULL, "ripples");
        Painter_CMD(&Leaf_A, NULL, NULL, NULL, "lotus_leaf");
        Painter_CMD(&Leaf_B, NULL, NULL, NULL, "lotus_leaf");
        Painter_CMD(&Leaf_C, NULL, NULL, NULL, "lotus_leaf");
        Painter_CMD(&Leaf_D, NULL, NULL, NULL, "lotus_leaf");
        Painter_CMD(&Leaf_E, NULL, NULL, NULL, "lotus_leaf");
        Painter_CMD(&Leaf_F, NULL, NULL, NULL, "lotus_leaf");
        Painter_CMD(NULL, NULL, NULL, Sprays, "sprays");
        Painter_CMD(Rain, &wind, Ripples, Sprays, "rain");

        Painter_Update(15);
    }

    Painter_End(Rain, Ripples, Sprays);
}
```

效果展示

见视频"rain_and_river.mp4"