Organ-Field GUI Help

0.1 概述

C++程序的交互界面为控制台(命令行),而对于初学者而言(或非计算机专业的同学),一个图形用户界面并不是必要的但的确能明显的提升程序的交互体验. 这一类型的群体自己编写图形界面或使用 C++的 GUI 库(如 QT)或多或少都觉得力不从心,从而难以兼顾程序核心功能与图形界面. 于是就诞生了基于很多初学者都在使用的 EasyX Graphics Library 的 GUI 库(Organ-Field GUI), 其最大的特点就是简单易用.初学者可以用其进行学习前期的过度,非计算机专业的同学可以把大部分精力投入于程序核心功能中的同时也能兼顾友好的交互.

Chapter 1 Organ-Field Button(OFB)

1.0 引用与类

目前 Organ-Field GUI 没有开发完成,因此在使用前需手动在你的项目中添加"OFButton.cpp"与"OFButton.h"文件.

使用 Organ-Field Button 前需要引用头文件: `#include "OFButton.h"`

使用 Organ-Field Button 前需要引用类名: `OFButton OFB;`

1.1 宏

- **在创建按钮时,需要指定按钮的状态,按钮共有 5 种状态,以下是按钮状态宏.**

宏 Macro	值 Value	含义 Meaning
normal	0	正常状态: 表示按钮是可交互的，并且可用的
disable	1	禁用状态: 表示当前组件处于非交互状态，但是之后可以被启用
hover	2	悬停状态: 当用户使用光标或者其他的元素，置于其上方的时候，显示这样的状态(鼠标划过)
click	3	激活状态: 表示用户已经按下按钮,且还未结束按按钮的动作(鼠标点击)
loading	4	加载状态: 表示操作正在加载中，组件正在反映，但是操作还未完成

1.2 函数

1.2.1 CreateButton

- **这个函数用于创建一个按钮**

```
int CreateButton(
    string ButtonName,
    int x, int y,
    int width, int height,
    short SetState,
    int (*EventFunc)()
);
```

- **参数**
  - < ButtonName > 设置按钮名称
  - < x,y > 指定按钮坐标
  - < width > 设置按钮宽度
  - < height > 设置按钮高度
  - < SetSate > 设置按钮状态(此处填写按钮状态宏)
  - < *EventFunc > 指定要执行的事件函数指针(在此填写函数名)

1.2.2 ButtonDefaultStyle

- **按钮样式快速设置函数,此函数令所有按钮使用同一套已经预设好的样式,不再需要自己进行复杂的按钮样式设置.**

```
void ButtonDefaultStyle();
```

1.2.3 SetButtonStyle Series

- **这一系列的函数用于自定义设置各按钮状态的样式(不能与 1.2.2 ButtonDefaultStyle 同时使用)**

```
//设置正常状态按钮的样式
void SetNormalStyle(
    short buttonrim,
    COLORREF rimcolor,
    short fillstyle,
    short fillhatch,
    COLORREF fillcolor,
    COLORREF textcolor
);
```

```
//设置禁用状态按钮的样式
void SetDisableStyle (
    short buttonrim,
    COLORREF rimcolor,
    short fillstyle,
    short fillhatch,
    COLORREF fillcolor,
    COLORREF textcolor
);
```

```
//设置悬停状态按钮的样式
void SetHoverStyle (
    short buttonrim,
    COLORREF rimcolor,
    short fillstyle,
    short fillhatch,
    COLORREF fillcolor,
    COLORREF textcolor
);
```

```
//设置激活状态按钮的样式
void SetClickStyle (
    short buttonrim,
    COLORREF rimcolor,
    short fillstyle,
    short fillhatch,
    COLORREF fillcolor,
    COLORREF textcolor
);
```

```
//设置加载状态按钮的样式
void SetLoadingStyle (
    short buttonrim,
    COLORREF rimcolor,
    short fillstyle,
    short fillhatch,
    COLORREF fillcolor,
    COLORREF textcolor
);
```

- 参数

```
< buttonrim >   设置按钮边框线粗细
< rimcolor >    设置按钮边框线颜色
< fillstyle >   设置按钮填充样式
< fillhatch >   指定填充图案
< fillcolor >   指定填充颜色
< textcolor >   指定按钮字体颜色
```

- 颜色

```
COLORREF RGB(
    BYTE byRed,     //颜色的红色部分
    BYTE byGreen,   //颜色的绿色部分
    BYTE byBlue     //颜色的蓝色部分
);
//颜色值范围: 0~255
```

- 填充样式

宏 Macro	含义 Meaning
BS_SOLID	固实填充
BS_NULL	不填充
BS_HATCHED	图案填充
BS_PATTERN	自定义图案填充
BS_DIBPATTERN	自定义图像填充

- 填充图案

宏 Macro	含义 Meaning
HS_HORIZONTAL	填充横线
HS_VERTICAL	填充竖线
HS_FDIAGONAL	填充斜线
HS_BDIAGONAL	填充反斜线
HS_CROSS`	填充网格线
HS_DIAGCROSS	填充斜网格线

- 示例代码

```
#include "Window.h"
#include "OFButton.h"
Window win;
OFButton OFB;

int main() {
    //初始化绘图窗口(窗口宽度,窗口高度,显示控制台);
    win.Initialize_Window(1200, 700, EW_SHOWCONSOLE);
    //设置各按钮状态的样式(按钮边框线粗细,边框线颜色,指定填充样式,指定填充图案,指定填充颜色,指定按钮字体颜色);
    OFB.SetNormalStyle(1, RGB(180, 180, 180), BS_SOLID, NULL, RGB(225, 225, 225), RGB(30, 30, 30));
    OFB.SetDisableStyle(1, RGB(191, 191, 191), BS_SOLID, NULL, RGB(204, 204, 204), RGB(130, 130, 130));
    OFB.SetHoverStyle(1, RGB(180, 180, 180), BS_SOLID, NULL, RGB(229, 243, 255), RGB(30, 30, 30));
    OFB.SetClickStyle(1, RGB(10, 89, 247), BS_SOLID, NULL, RGB(204, 232, 255), RGB(30, 30, 30));
    OFB.SetLoadingStyle(1, RGB(10, 89, 247), BS_SOLID, NULL, RGB(204, 204, 204), RGB(130, 130, 130));
    system("pause");
};
//注:  在不需要参数时必须填 NULL;
```

1.3 示例

- 以下代码展示如何创建两个按钮

```
//编译环境:  Visual Studio 2022   C++14   EasyX_20220116   使用多字节字符集;
#include "Window.h"
#include "OFButton.h"
Window win;
OFButton OFB;

void ClickEvent() { cout << "ButtonClick-1" << endl; } //第一个按钮点击事件;
void ClickEvent2() { cout << "ButtonClick-2" << endl; }//第二个按钮点击事件;

int main() {
    win.Initialize_Window(1200, 700, EW_SHOWCONSOLE); //初始化窗口(窗口宽度:1200px,窗口高度:700px,显示控制台);

    OFB.ButtonDefaultStyle();//快速设置按钮样式为预设默认值;
    while (1) {//这里一定要给循环;
        OFB.CreateButton("Test", 10, 10, 120, 60, normal, ClickEvent);   //创建第一个按钮;
        OFB.CreateButton("Test2", 10, 80, 120, 60, normal, ClickEvent2); //创建第二个按钮;
    }
};
```

Chapter 2 Organ-Field Window(OFW)

2.0 引用与类

目前 Organ-Field GUI 没有开发完成,因此在使用前需手动在你的项目中添加"Window.cpp"与"Window.h"文件.

使用 Organ-Field Window 前需要引用头文件: `#include "Window.h"`

使用 Organ-Field Window 前需要引用类名: `Window win`

2.1 函数

2.1.1 Initialize_Window

- **这个函数用于创建窗口(内置默认窗口样式)**

```
void Initialize_Window(int Width, int Height, int Flag);
```

- **参数**

< Width > 指定窗口宽度

< Height > 指定窗口高度

< Flag > 绘图窗口的样式

- **绘图窗口的样式宏(Flag Macro)**

宏 Macro	含义 Meaning
EW_DBLCLKS	在绘图窗口中支持鼠标双击事件
EW_NOCLOSE	禁用绘图窗口的关闭按钮
EW_NOCLOSE	禁用绘图窗口的最小化按钮
EW_NOCLOSE	显示控制台窗口

2.1.2 SetWindowStyle

- **这个函数用于设置窗口样式,必须在 2.1.1 Initialize_Window 创建窗口后才可使用**

```
void SetWindowStyle(COLORREF backdrop_color, float xasp, float yasp);
```

- **参数**

< backdrop_color > 指定窗口背景颜色

< xasp > x 方向上的缩放因子(例如绘制宽度为 100 的矩形，实际的绘制宽度为 100 * xasp)

< yasp > y 方向上的缩放因子(例如绘制高度为 100 的矩形，实际的绘制高度为 100 * yasp)

- **备注**

① 如果缩放因子为负，可实现坐标轴翻转.

② 使用函数时为了更新背景颜色,此函数会清空一次窗口绘图画板.

2.1.3 SetWindowTilte

- **这个函数用于设置窗口标题**

```
void SetWindowTilte(const string &Title);
```

- **参数**

< &Title > 指定窗口标题(文本)

2.1.4 MoveWindow

- **这个函数用于设置窗口位置**

```
void MoveWindow(int X, int Y);
```

- **参数**

< X > 相对屏幕的横轴坐标

< Y > 相对屏幕的纵轴坐标

- **备注**

屏幕的最左上角为坐标原点,向下纵坐标增加,向右横坐标增加(此法则也适用于窗口).

2.1.5 Reset_Window

- **这个函数用于重新设置窗口大小并清空一次窗口绘图画板**

```
void Reset_Window(int Width, int Height);
```

- **参数**

< Width > 指定新的窗口宽度

< Height > 指定新的窗口高度

2.1.6 WindowOnTop

- 这个函数用于使窗口置顶

```
void WindowOnTop();
```

2.1.7 GetWindowHWND

- 这个函数用于获取窗口句柄

```
HWND GetWindowHWND();
```

- 返回值

返回本窗口的窗口句柄(hWnd)

2.1.8 GetWindowRect

- 这个函数用于获取窗口边框矩形

```
RECT GetWindowRect();
```

- 返回值

返回本窗口的窗口边框矩形

```
typedef struct tagRECT{
    LONG left;    //窗口矩形左侧边框;
    LONG top;     //窗口矩形顶部边框;
    LONG right;   //窗口矩形右侧边框;
    LONG bottom;  //窗口矩形底部边框;
} RECT;
```

- 备注

这里的矩形有两种表达方法: ①矩形的四边; ②矩形的左上角与右下角点的坐标;

2.1.9 GetWidth

- 这个函数用于获取窗口宽度

```
static int GetWidth();
```

- 返回值

返回本窗口的窗口宽度

2.1.10 GetHeight

- 这个函数用于获取窗口高度

```
static int GetHeight();
```

- 返回值

返回本窗口的窗口高度

Chapter 3 Organ-Field Message (OFM)

3.0 引用与类

目前 Organ-Field GUI 没有开发完成,因此在使用前需手动在你的项目中添加"OFMessage.cpp"与"OFMessage.h"文件.

使用 Organ-Field Window 前需要引用头文件: `#include "OFMessage.h"`

使用 Organ-Field Window 前需要引用类名: `OFMessage OFM;`

3.1 函数

3.1.1 PeekOFMessageUP

- 这个函数用于启动一个不断扫描鼠标消息(包括位置,按键,滚轮)和键盘消息以及窗口消息的独立函数线程.

```
void PeekOFMessageUP();
```

- 备注

这个函数负责启动一个独立于主线程的线程,然后将扫描得到的消息储存于 ExMessage 结构体中,需要鼠标位置等消息时直接从结构体中调取,不会卡死主线程.

3.1.2 PeekOFMessage

- 这个函数用于获取鼠标消息(包括位置,按键,滚轮)和键盘消息以及窗口消息.

```
ExMessage PeekOFMessage();
```

- 返回值

立即返回保存有鼠标消息和键盘消息以及窗口消息的 ExMessage 结构体.

```
struct ExMessage{
     USHORT message; // The message identifier.
     union{// Data of the mouse message
          struct{
               bool ctrl   :1; // Indicates whether the CTRL key is pressed.
               bool shift  :1; // Indicates whether the SHIFT key is pressed.
               bool lbutton:1; // Indicates whether the left mouse button is pressed.
               bool mbutton:1; // Indicates whether the middle mouse button is pressed.
               bool rbutton:1; // Indicates whether the right mouse button is pressed.
               short x;    // The x-coordinate of the cursor.
               short y;    // The y-coordinate of the cursor.
               short wheel;// The distance the wheel is rotated, expressed in multiples or divisions of 120.
          };
          struct{// Data of the key message
               BYTE vkcode;// The virtual-key code of the key.
               BYTE scancode; // The scan code of the key. The value depends on the OEM.
               bool extended:1;
               // Indicates whether the key is an extended key, such as a function key or a key on the numeric key
               pad. The value is true if the key is an extended key; otherwise, it is false.
               bool prevdown:1; // Indicates whether the key is previously up or down.
          };
          TCHAR ch; // Data of the char message
```

```
        struct{// Data of the window message
            WPARAM wParam;
            LPARAM lParam;
        };
    };
};
```

- **备注**

此函数只会返回 PeekOFMessageUP 线程预先储存好的消息,对性能几乎没有影响.

3.3 示例

- **以下代码展示如何获取鼠标坐标**

```
//编译环境：Visual Studio 2022  C++14  EasyX_20220116  使用多字节字符集;
Window OFW;
OFMessage OFM;


ExMessage MouseMessage{ NULL };  //定义消息变量(它能够储存鼠标的坐标);
int main() {
    OFW.Initialize_Window(1200, 700, EW_SHOWCONSOLE);

    OFM.PeekOFMessageUP();//启动鼠标消息(坐标)扫描;


    while (true) {
        MouseMessage = OFM.PeekOFMessage()//获取鼠标信息,并将返回值传给 MouseMessage;

        //打印鼠标坐标值;
        cout << "鼠标当前坐标(X:" << MouseMessage.x << ",Y:" << MouseMessage.y << ")" << endl;
    }
};
```

Chapter 4 Organ-Field ToggleSwitch (OFTS)

4.0 引用与类

目前 Organ-Field GUI 没有开发完成,因此在使用前需手动在你的项目中添加"ToggleSwitch.cpp"与"ToggleSwitch.h"文件.

使用 Organ-Field Window 前需要引用头文件: `#include "ToggleSwitch.h"`

使用 Organ-Field Window 前需要引用类名: `ToggleSwitch OFTS;`

4.1 函数

4.1.1 CreateToggle

- **这个函数用于创建一个拨动开关**

```
int CreateToggle(int ToggleName, int x, int y, short SetInitialState, int (*EventFunc)());
```

- **参数**

< ToggleName > 设置拨动开关名称(此名称只能是非负整数数值,并且应遵从由小到大取名;如:0,1,2,3,4,……)
< x,y > 指定拨动开关坐标
< width > 设置拨动开关宽度
< height > 设置拨动开关高度
< SetSate > 设置拨动开关状态(此处填写拨动开关状态宏)
< *EventFunc > 指定要执行的事件函数指针(在此填写函数名)

- **拨动开关状态宏**

宏 Macro	含义 Meaning
open	拨动开关打开
close	拨动开关关闭
disable	拨动开关禁用
loading	正在加载: 拨动开关被拨动后所执行的事件函数工作未完成

- **返回值**

返回事件函数的返回值.

4.1.2 SetToggleState

- **这个函数用于设置拨动开关的状态.**

```
void SetToggleState(int ToggleName, short SetState);
```

- **参数**

< ToggleName > 指定拨动开关名称(此名称只能是非负整数数值)
< SetState > 指定拨动开关的状态(此处填写拨动开关状态宏)

4.1.3 GetToggleState

- **这个函数用于获取拨动开关的状态.**

```
short GetToggleState(int ToggleName);
```

- **参数**

< ToggleName > 指定拨动开关名称(此名称只能是非负整数数值)

- **返回值**

返回目标拨动开关的状态宏

4.1.4 ToggleDefaultStyle

- **拨动开关样式快速设置函数,此函数令所有拨动开关使用同一套已经预设好的样式,不再需要自己进行复杂的样式设置.**

```
void ToggleDefaultStyle();
```

4.1.5 SetToggleStyle Series

- **这一系列的函数用于自定义设置各拨动开关状态的样式(不能与 4.1.4 ToggleDefaultStyle 同时使用)**

4.1.5.1 SetOpenStyle

- **设置拨动开关打开状态的样式**

```
void SetOpenStyle(COLORREF fillcolor, COLORREF slidercolor);
```

- **参数**

< fillcolor > 指定拨动开关填充底色
< slidercolor > 指定拨动开关滑块颜色

4.1.5.2 SetCloseStyle

- **设置拨动开关关闭状态的样式**

```
void SetCloseStyle(short Togglerim, COLORREF rimcolor, COLORREF slidercolor);
```

- **参数**

< Togglerim > 指定拨动开关边框线粗细(px)
< rimcolor > 指定拨动边框线颜色
< slidercolor > 指定拨动开关滑块颜色

4.1.5.3 SetDisableStyle

- **设置拨动开关禁用状态的样式**

```
void SetDisableStyle(short Togglerim, COLORREF rimcolor, COLORREF slidercolor);
```

- **参数**

< Togglerim > 指定拨动开关边框线粗细(px)
< rimcolor > 指定拨动边框线颜色
< slidercolor > 指定拨动开关滑块颜色

4.1.5.4 SetLoadingStyle

- **设置拨动开关加载状态的样式**

```
void SetLoadingStyle(short Togglerim, COLORREF rimcolor, COLORREF slidercolor);
```

- **参数**

< Togglerim > 指定拨动开关边框线粗细(px)
< rimcolor > 指定拨动边框线颜色
< slidercolor > 指定拨动开关滑块颜色

4.1.5.5 SetHoverStyle

- **设置当鼠标悬停在拨动开关上方时的状态样式**

```
void SetHoverStyle(short Togglerim, COLORREF MainColor);
```

- **参数**

< Togglerim > 指定拨动开关边框线粗细(px)
< MainColor > 指定鼠标划过时的主题色

4.2 示例

- **以下代码展示如何创建拨动开关**

```
//编译环境：Visual Studio 2022  C++14  EasyX_20220116  使用多字节字符集;
#include "Universal.h"
#include "Window.h"
#include "ToggleSwitch.h"
Window OFW;
ToggleSwitch TS;
OFMessage OFM;

int ToggleEvent1() { cout << "ToggleOpen-1" << endl; return 0; }
int ToggleEvent2() { cout << "ToggleOpen-2" << endl; return 0; }

int main() {
    OFW.Initialize_Window(400, 400, EW_SHOWCONSOLE);
    TS.ToggleDefaultStyle();//快速设置拨动开关样式;

    OFM.PeekOFMessageUP();//启动鼠标消息(坐标)扫描;

    while (1) {
        TS.CreateToggle(0, 10, 40, close, ToggleEvent1);   //创建第一个拨动开关;
        TS.CreateToggle(1, 10, 70, disable, ToggleEvent2); //创建第二个拨动开关;
    }
};
```

Chapter 5 Organ-Field Slider (OFS)

5.0 引用与类

目前 Organ-Field GUI 没有开发完成,因此在使用前需手动在你的项目中添加"OFSlider.cpp"与"OFSlider.h"文件.

使用 Organ-Field Window 前需要引用头文件: `#include "OFSlider.h"`

使用 Organ-Field Window 前需要引用类名: `OFSlider OFS;`

5.1 函数

5.1.1 CreateSlider

- **这个函数用于创建一个滑动条或进度条**

```
float CreateSlider(int SliderName, int x, int y, int width, int height, short SetState);
```

- **参数**

< SliderName > 设置滑动条或进度条名称(此名称只能是非负整数数值,并且应遵从由小到大取名;如:0,1,2,3,4,……)
< x,y > 指定滑动条或进度条坐标
< width > 设置滑动条或进度条宽度
< height > 设置滑动条或进度条高度
< SetSate > 设置滑动条或进度条状态(此处填写滑动条或进度条状态宏)

- **滑动条或进度条状态宏**

宏 Macro	含义 Meaning
normal	滑动条正常状态(可拖动)
disable	滑动条禁用拖动(仅作为进度条)

- **返回值**

返回滑动条或进度条当前进度位置(0~100)

5.1.2 SetSliderLocation

- **这个函数用于设置滑动条或进度条进度**

```
void SetSliderLocation(int SliderName, float LocationValue);
```

- **参数**

< SliderName > 指定滑动条或进度条名称(此名称只能是非负整数数值)
< LocationValue > 设置滑动条或进度条进度位置(0~100)

5.1.3 PeekSliderLocation

- **这个函数用于获取当前滑动条或进度条进度**

```
float PeekSliderLocation(int SliderName);
```

- **参数**

< SliderName > 指定滑动条或进度条名称(此名称只能是非负整数数值)

- **返回值**

返回目标滑动条或进度条进度位置(0~100)

5.1.4 SliderDefaultStyle

- **滑动条或进度条样式快速设置函数,使用同一套已经预设好的样式,不再需要自己进行复杂的样式设置.**

```
void SliderDefaultStyle ();
```

5.1.5 SetSliderStyle Series

- **这一系列的函数用于自定义设置各滑动条状态的样式(不能与 5.1.4 SliderDefaultStyle 同时使用)**

5.1.5.1 SetNormalStyle

- **设置滑动条或进度条正常状态状态的样式**

```
void SetNormalStyle(COLORREF rimcolor, COLORREF fillcolor, COLORREF bkcolor, COLORREF TextColor);
```

- **参数**

< rimcolor > 边框线粗细(px)
< fillcolor > 边框线颜色
< bkcolor > 滑动条或进度条填充底色
< TextColor > 进度字体颜色

5.1.5.2 SetDisableStyle

- **设置滑动条或进度条禁用状态状态的样式**

```
void SetDisableStyle(COLORREF rimcolor, COLORREF fillcolor, COLORREF bkcolor, COLORREF TextColor);
```

- **参数**

< rimcolor > 边框线粗细(px)
< fillcolor > 边框线颜色
< bkcolor > 滑动条或进度条填充底色
< TextColor > 进度字体颜色

5.1.5.3 SetHoverStyle

- **设置当鼠标悬停在滑动条或进度条上方时的状态样式**

```
void SetHoverStyle(COLORREF rimcolor, COLORREF fillcolor, COLORREF bkcolor, COLORREF TextColor);
```

- **参数**

< rimcolor > 边框线粗细(px)
< fillcolor > 边框线颜色
< bkcolor > 滑动条或进度条填充底色
< TextColor > 进度字体颜色

5.1.5.4 SetDragStyle

- **设置滑动条被拖动状态的样式**

```
void SetDragStyle (COLORREF rimcolor, COLORREF fillcolor, COLORREF bkcolor, COLORREF TextColor);
```

- **参数**

< rimcolor > 边框线粗细(px)
< fillcolor > 边框线颜色
< bkcolor > 滑动条或进度条填充底色
< TextColor > 进度字体颜色

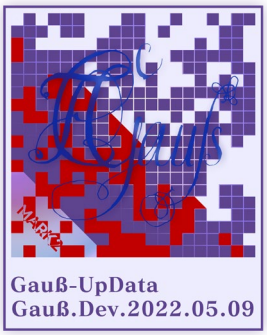
5.2 示例

- **以下代码展示如何创建进滑动条和度条**

```
//编译环境：Visual Studio 2022  C++14  EasyX_20220116  使用多字节字符集;
#include "Universal.h"
#include "Window.h"
#include "OFMessage.h"
#include "OFSlider.h"
Window OFW;
OFMessage OFM;
OFSlider OFS;

int main() {
    OFW.Initialize_Window(400, 400, EW_SHOWCONSOLE);
    OFS.SliderDefaultStyle();//快速设置拨动开关样式;
    OFS.SetSliderLocation(0, 0); //设置滑动条初始进度为 0;
    OFS.SetSliderLocation(1, 0); //设置进度条初始进度为 0;
    OFM.PeekOFMessageUP();//启动鼠标消息(坐标)扫描;
    while (1) {
        OFS.CreateSlider(0, 10, 360, 380, 30, normal);  //创建一个滑动条;
        OFS.CreateSlider(1, 10, 360, 380, 30, disable); //创建一个进度条;
    }
};
```

Chapter 6 Organ-Field Blur (OFBlur)



6.0 引用与类

目前 Organ-Field GUI 没有开发完成,因此在使用前需手动在你的项目中添加"OFBlur.cpp"与"OFBlur.h"文件.

使用 OFBlur 前需要引用头文件: `#include "OFBlur.h "`

使用 OFBlur 前需要引用类名: `OFBlur OFBr;`

6.1 public 函数

6.1.1 WriteBlurCache

- **这个函数用于将原始图像写入模糊缓存(相当于截图)**

```
void WriteBlurCache(int BlurName, int x, int y, int width, int height);
```

- **参数**

< BlurName > 指定模糊名称(只能设置非负整形值)
< x,y > 指定模糊坐标
< width > 设置模糊宽度
< height > 设置模糊高度

6.1.2 ColorFilter

- **线性颜色滤镜使原始图像的颜色往设定值颜色偏移(该函数用于操作模糊缓存中的原始图像)**

```
void ColorFilter(int BlurName, int R, int G, int B);
```

- **参数**

< BlurName > 指定模糊名称(只能指定非负整形值)
< R,G,B > 指定 RGB 颜色偏移系数(可为任何整形值,具体效果请自行尝试)

- **附加 CPP 内部色彩偏移算法(方便理解)**

```
DWORD *pMem = GetImageBuffer(&IMG[BlurName].ColorFilterIMG);
float WeightR = NULL, WeightG = NULL, WeightB = NULL;
for (int i = 0; i < IMG[BlurName].OrigIMG.getwidth() * IMG[BlurName].OrigIMG.getheight(); ++i) {
float r = (GetRValue(pMem[i]) + static_cast<float>(R) * 0.5); if (r > 255) { r = 255; } else if (r < 0) { r = 0; }
float g = (GetGValue(pMem[i]) + static_cast<float>(G) * 0.5); if (g > 255) { g = 255; } else if (g < 0) { g = 0; }
float b = (GetBValue(pMem[i]) + static_cast<float>(B) * 0.5); if (b > 255) { b = 255; } else if (b < 0) { b = 0; }
pMem[i] = RGB((BYTE)r, (BYTE)g, (BYTE)b);
}
```

6.1.3 CreateGaussBlur

- **创建高斯模糊效果(该函数会创建一个独立于程序主线程之外的线程用于高斯模糊计算)**

```
void CreateGaussBlur(int BlurName, const int radius, const double SigmaDis);
```

- **参数**

< BlurName > 指定模糊名称(只能指定非负整形值)
< radius > 模糊半径(模糊半径越大模糊效果越明显,计算速度越慢)
< SigmaDis > Sigma 值越小模糊质量越好,计算速度越慢

- **性能说明**

由于本人 C++编程能力实在太渣,所以写不出高性能的高斯模糊算法,因此 OFBlur 的模糊功能存在较为严重的性能问题,计算速度比较慢无法进行实时模糊处理,小面积,小半径的高斯模糊计算速度其实可以接受.不过我也做了多线程优化,并且所有的模糊计算都是独立于程序主线程之外的,经过我的暴力测试最多可以使用 400(左右)个线程来进行高斯模糊计算,可以完全榨干 6 核 12 线程的 R5-4600H CPU 的性能.总之不建议过多的使用模糊效果.

6.1.4 ShowBlur

- **显示模糊(将已经计算完成的模糊图像显示出来)**

```
void ShowBlur(int BlurName, float ALPHA);
```

- **参数**

< BlurName > 指定模糊名称(只能指定非负整形值)
< ALPHA > 不透明度(0:完全透明~1:完全不透明)

- **说明**

此函数只会显示已经全部计算完成的高斯模糊效果,若模糊计算未完成,则该函数会等待计算完成后再进行显示;该函数的线程同样也是独立于程序主线程之外的,该函数的等待不会影响主线程.

6.1.5 ShowOrig

- **显示模糊前原始图像**

```
void ShowOrig (int BlurName, float ALPHA);
```

- **参数**

< BlurName > 指定模糊名称(只能指定非负整形值)
< ALPHA > 不透明度(0:完全不透明~1:完全透明 000)

6.2 “模糊缓存(BlurCache)”的结构

```
typedef struct BlurCache {
      IMAGE OrigIMG; unsigned int x; unsigned int y;                  //原始图片缓存;
      IMAGE ColorFilterIMG;                                           //原始图片滤色后的缓存;
      IMAGE BlurIMG; int radius; double SigmaDis; double SigmaCol;    //模糊后的图片缓存;
      IMAGE ShowIMG;                                                    //最终显示模糊的图片缓存;
      bool BlurDone;                                                    //模糊运算是否完成;
}BlurCache;
```

6.3 示例

- **以下代码展示如何创建模糊.** //编译环境：Visual Studio 2022 C++14 EasyX_20220116 使用多字节字符集;

```
#include "Universal.h"
#include "Window.h"
#include "OFMessage.h"
#include "OFBlur.h"
Window win;
OFMessage OFM;
OFBlur OFBr;

int main() {
      win.Initialize_Window(800, 800, EW_SHOWCONSOLE);
      win.SetWindowTilte("Organ-Field GUI");

      loadimage(NULL, _T("C:\\Test.png"), 800, 800);        //从硬盘读取并显示原始图片;

      OFBr.WriteBlurCache(0, 0, 0, 800, 800);               //将原始图片缓存到 BlurCache 中;
      OFBr.ColorFilter(0, -130, -130, -130);                //对原始图片进行色彩偏移处理;
      OFBr.CreateGaussBlur(0, 40, 20);                      //对色彩偏移后的图像计算高斯模糊;
      OFBr.ShowBlur(0, 1);                                  //模糊计算完成后立即显示模糊效果;

      system("pause");
};
```

Chapter 7 Organ-Field TaskManager (OFTM)

RMSHE
2022.07.17
14:05:30 +08'00'
2021.001.20138

7.0 引用与类

目前 Organ-Field GUI 没有开发完成,因此在使用前需手动在你的项目中添加"OFTools.cpp"与"OFTools.h"文件.

使用 OFTM 前需要引用头文件: `#include "OFTools.h "`

使用 OFTM 前需要引用类名: `OFTaskManager OFTM;`

7.1 public 函数

7.1.1 ProcessKill

- **这个函数用于结束指定进程**

```
void ProcessKill(LPCTSTR strProcessName);
```

- **参数**

< strProcessName > 目标进程名称

- **备注**

此函数为非主线程堵塞函数;

7.1.2 ProcessTraversal

- **这个函数用于获取全部进程的列表**

```
void ProcessTraversal();
```

- **备注**

此函数为非主线程堵塞函数;

7.1.3 GetTaskManagerResult

- **这个函数用于调用(获取)在 TaskManagerType 中保存的相关信息**

```
TaskManagerType GetTaskManagerResult();
```

- **返回值**

返回 TaskManagerType TMT;

- **说明**

ProcessKill 函数的返回值[1.要结束的目标进程名称],[2.结束进程的结果(进程是否被结束)]; 以及通过 ProcessTraversal 获取到的进程列表均保存在 TaskManagerType 类型中,此函数提供了一个外部接口来调用这些信息.

7.2 “TaskManagerType”的结构

```
typedef struct TaskManagerType {
      string Processlist;              //系统进程列表;
      LPCTSTR Target_strProcessName;   //要结束的目标进程名称;
      BOOL ProcessKill_Result;         //结束进程的结果(进程是否被结束);
}TaskManagerType;
```

TaskManagerType (TaskManager 类型)是保存 TaskManager 类中所有成员信息的数据类型,其中外部有两种调用这些信息的方式: 一是通过 GetTaskManagerResult 函数; 二是因为 TaskManagerType TMT 是 public 中的成员变量,故可以通过类似 OFTM.TMT.Processlist 的方式调用.

7.3 示例

- **以下代码展示获取进程列表并且结束一个进程.** //编译环境: Visual Studio 2022 C++14 EasyX_20220116 使用多字节字符集;

```
#include "Universal.h"
#include "OFTools.h"
OFTaskManager OFTM;

int main() {
      OFTM.ProcessTraversal();                  //获取系统进程列表;
      OFTM.ProcessKill("Taskmgr.exe");          //结束"Taskmgr.exe(任务管理器)"的进程;

      //这里是以两种不同的方式获取 TaskManagerType 中 ProcessKill_Result 与 Processlist(进程列表)的信息;
      cout << OFTM.TMT.ProcessKill_Result << "\n" << OFTM.TMT.Processlist << endl;
      cout << OFTM.GetTaskManagerResult().ProcessKill_Result << "\n" << OFTM.GetTaskManagerResult().Processlist
      << endl;

      system("pause");
};
```