# u=2128609852,3617731533&fm=0&gp=0

**数据结构课程设计实验报告**

**细胞图像处理**

姓名：张晋恺

学号：201626810926

班级：软工1602

完成日期： 2017/1/8

目录

[**[实验题目和要求] 3**](#_Toc12344)

[**1. 对于输入的测试图像，能够输出正确的填充结果 3**](#_Toc31251)

[**2. 要求采用Visual studio 2010及以上版本进行调试； 3**](#_Toc16069)

[**[设计思路] 4**](#_Toc14656)

[**一、系统总体设计 4**](#_Toc25680)

[**二、 系统功能设计 4**](#_Toc30472)

[**三、 算法设计思路 5**](#_Toc6356)

[**四、 类的设计 5**](#_Toc6356)

[**五、 主程序的设计 6**](#_Toc9350)

[**[调试分析] 7**](#_Toc29019)

1. **技术难点 7**
2. **调试错误分析 8**

[**[测试结果] 10**](#_Toc12683)

[**一、图像读取，保存 10**](#_Toc22941)

[**二、图像更新，细胞上色处理 11**](#_Toc20850)

[**[附录] 12**](#_Toc23556)

[**1. MainFrm.h 12**](#_Toc24464)

[**2. painter.h 12**](#_Toc10283)

[**3. painterDoc.h 13**](#_Toc18402)

[**4. painterView.h 18**](#_Toc16924)

[**5. MainFrm.cpp 19**](#_Toc16898)

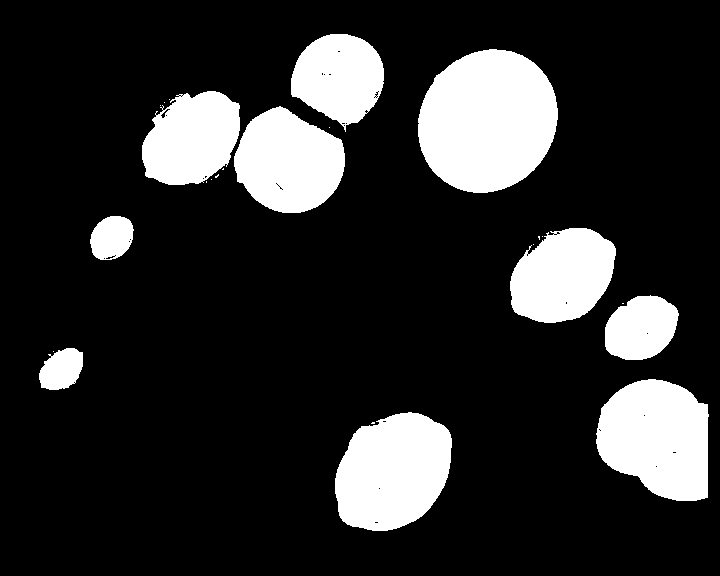
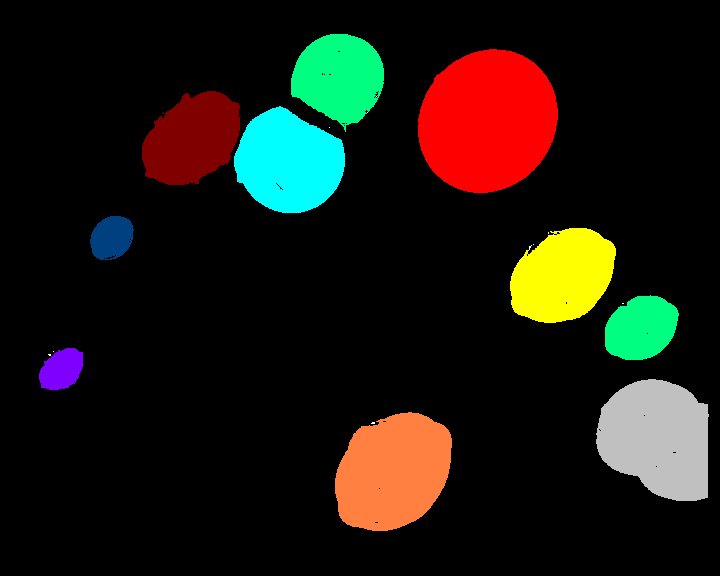
[**6. painter.h 20**](#_Toc24395)

[**7. painterDoc.cpp 21**](#_Toc31316)

[**8. painterView.h 24**](#_Toc10793)

# [实验题目和要求]

**【问题描述】**在细胞图像处理和识别中，区域特征是非常重要的，也是分析细胞特征的前提和基础。这些区域特征可以是外接矩形、像素点个数等。因此，我们需要得到跟踪得到不同区域，才能对其进行特征分析。针对该问题，我们需要设计一个类和相关方法，用于跟踪得到每个区域，并填上不同的颜色。

** **

**【基本要求】**

1. 对于输入的测试图像，能够输出正确的填充结果
2. 要求采用Visual studio 2010及以上版本进行调试；

【**实现提示**】

1. 已经有的类处理函数：图像读取和写入函数; 图像中某一点的像素值读取和更新; 不同颜色的值表示; 图像处理代码示例(具体见附件中的代码, 即压缩包文件BitmapAPP.rar)。
2. 可以采用STL模板库

**【运行结果要求】**实验报告要求有详细的设计思路、功能测试截图。给出区域跟踪和上色的类结构，并对类成员和方法进行说明和解释。给出关键算法的流程图。给出调试过程中三个或三个以上的问题和解决方法。

**【考核要求】**要求程序能正常运行，全面完成题目要求。

**【题目难度】 难，成绩等级次高**

**【咨询教师】可以咨询任课教师，也可咨询潘翔老师(Email:** [**panx@zjut.edu.cn**](mailto:panx@zjut.edu.cn)**, QQ:252444679)**

## [设计思路]

一、系统总体设计

程序采用 vs2015 编写。建立的是MFC项目，微软基础类库（英语：Microsoft Foundation Classes，简称MFC）是微软公司提供的一个类库（class libraries），以C++类的形式封装了Windows API，并且包含一个应用程序框架，以减少应用程序开发人员的工作量。其中包含大量Windows句柄封装类和很多Windows的内建控件和组件的封装类。

在Windows平台下，GDI（Graphics Device Interface）图形设备接口被抽象为上下文CDC类（Device Context，DC）。Windows平台直接接收图形数据信息的不是显示器和打印机等硬件设备，而是CDC对象。MFC中，CDC类定义设备上下文对象的基类，封装了所需的成员函数，调用CDC类的成员函数，绘制和打印图形及文字。

简而言之，利用MFC对于图形的处理非常方便。

1. 系统功能设计

程序的功能设计由3部分组成：图像读取及保存，图像显示及更新，细胞上色处理，具体如下：

|  |  |  |
| --- | --- | --- |
| 功能 | 描述 | 效果演示 |
| 图像读取及保存 | 从文件目录中导入图像文件，  保存上色处理后的图像文件。 |  |
| 图像显示及更新 | 导入图像文件后将图像显示在窗口部件上，  在细胞上色过程后更新上色后的细胞图像 |  |
| 细胞上色处理 | 对导入的图像文件进行区域识别，上色填充等操作。  可以手动也可以自动 |  |

1. 算法设计思路

逐步读入图像，希望能设计出一个算法，识别出各自的白色的细胞区域，然后进行填充

1. 类的设计

程序主要5个类组成，CMainFrame，CpainterApp、CpainterDoc、CpainterView和CAboutDlg，它们的详细构成及相互关系如下：

。

1. CpainterApp: 定义了窗体加载之前需要做的预处理。
2. CMainFrame()：是程序的主窗口类，包含了对工具栏、状态栏、窗口的定义。
3. CAboutDlg：定义了ABOUT对话框
4. CpainterView: 包含了最主要的处理功能。

virtual void OnDraw(CDC\* pDC); // 每次点击完按钮后调用该函数，绘制视图

afx\_msg void OnFileOpen(); //点击打开按钮

afx\_msg void OnFileSave(); //点击保存按钮

afx\_msg void OnShowTwo(); //点击显示两张图片

afx\_msg void OnNext(); //点击手动上色

afx\_msg void OnStart(); //点击自动上色（两种上色都包含了区域跟踪和颜色更新，以自动上色为例展示）

{

if (numPicture == 0)

{

AfxMessageBox("载入图片后才能细胞上色!", MB\_OK, 0);

return;

}

AfxMessageBox("细胞上色!", MB\_OK, 0);

//打开临时的图片

FILE \*fpo = fopen(BmpName, "rb");

FILE \*fpw = fopen(BmpNameLin, "wb+");

//读取文件

fread(&bfh, sizeof(BITMAPFILEHEADER), 1, fpo);

fread(&bih, sizeof(BITMAPINFOHEADER), 1, fpo);

fwrite(&bfh, sizeof(BITMAPFILEHEADER), 1, fpw);

fwrite(&bih, sizeof(BITMAPINFOHEADER), 1, fpw);

unsigned char fred, fgreen, fblue;

}

fread(m\_pImage, m\_nImage, 1, fpo);

int white=1;

int count=0;

for(white;white>0;white--)

{

int first = 0;

int number = 0;

srand((int)time(0));

int RED = (rand() % 255), GREEN = (rand() % 255), BLUE = (rand() % 255);

for(int k=count;k>0;k--)

RED = (rand() % 255), GREEN = (rand() % 255), BLUE = (rand() % 255);

for (int i = 0; i < m\_nImage; i = i + 3)

{

fred = m\_pImage[i];

fgreen = m\_pImage[i + 1];

fblue = m\_pImage[i + 2];

if (fred == 255 && fgreen == 255 && fblue == 255 && first == 0)

{

count++;

white++;

m\_pImage[i] = RED;

m\_pImage[i + 1] = GREEN;

m\_pImage[i + 2] = BLUE;

first = 1;

}

if (m\_pImage[i] == RED && m\_pImage[i + 1] == GREEN && m\_pImage[i + 2] == BLUE)

{

int x1 = i + 3, x2 = i - 3, y1 = i + m\_nLineByte, y2 = i - m\_nLineByte;

if (m\_pImage[x1] == 255 && m\_pImage[x1 + 1] == 255 && m\_pImage[x1 + 2] == 255)

{

m\_pImage[x1] = RED;

m\_pImage[x1 + 1] = GREEN;

m\_pImage[x1 + 2] = BLUE;

number++;

}

if (m\_pImage[x2] == 255 && m\_pImage[x2 + 1] == 255 && m\_pImage[x2 + 2] == 255)

{

m\_pImage[x2] = RED;

m\_pImage[x2 + 1] = GREEN;

m\_pImage[x2 + 2] = BLUE;

i = i - 6;

number++;

}

if (m\_pImage[y1] == 255 && m\_pImage[y1 + 1] == 255 && m\_pImage[y1 + 2] == 255)

{

m\_pImage[y1] = RED;

m\_pImage[y1 + 1] = GREEN;

m\_pImage[y1 + 2] = BLUE;

number++;

}

if (m\_pImage[y2] == 255 && m\_pImage[y2 + 1] == 255 && m\_pImage[y2 + 2] == 255)

{

m\_pImage[y2] = RED;

m\_pImage[y2 + 1] = GREEN;

m\_pImage[y2 + 2] = BLUE;

number++;

}

}

}

if (number<300)

for (int i = 0; i < m\_nImage; i = i + 3)

{

fred = m\_pImage[i];

fgreen = m\_pImage[i + 1];

fblue = m\_pImage[i + 2];

if (fred == RED && fgreen == GREEN && fblue == BLUE)

{

m\_pImage[i] = 0;

m\_pImage[i + 1] = 0;

m\_pImage[i + 2] = 0;

}

}

}

fwrite(m\_pImage, m\_nImage, 1, fpw);//上色

fclose(fpo);

fclose(fpw);

level = 1;

numPicture = 2;

Invalidate();

void ShowBitmap(CDC\* pDC, CString BmpName); //显示位图函数

bool ReadBmp(); //用来读取bmp图片信息

bool SaveBmp(LPCSTR lpFileName); //用来保存bmp格式图片

1. 主程序的设计

程序功能实现及运行流程如下：

创建

对象并显示

等待用户操作

调用OnFileOpen()载入图像并显示

用户点击关闭按钮

程序退出

用户进行打开图片操作

用户进行保存图片操作

调用OnFileSave()保存图像

用户点击自动上色按钮

调用OnStart()自动上色

调用OnNext()

手动上色

用户点击手动按钮

用户点击显示预览窗按钮

程序显示预览窗

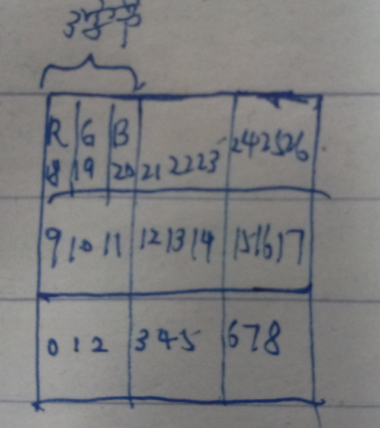
## [调试分析]

1. 技术难点：

单纯对细胞上色是很简单的，对于一幅位图，背景色为黑色，细胞为白色。给所有细胞上色只需要遍历算有像素点，对于白色的像素点，将它替换为任意想要的颜色即可，然后上色即可了。

**难点一：**如何遍历像素点，由于以前只尝试过读写txt文件，这次读写的是bmp位图文件，那么如何表示表示像素点是个问题，只有明白像素点如何读入，才能进行后续所有操作。

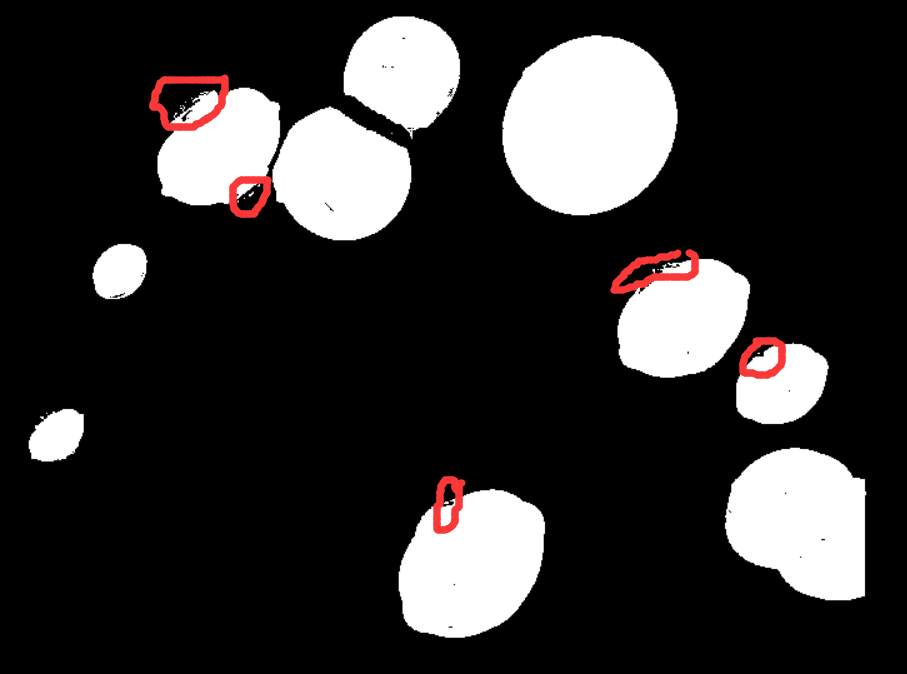
解决：经过了解，同样可以是通过指针读写，一个像素点的读写需要指针移动三次，因为需要读入RED, BLUE, GREEN也即RGB值。

如图：

由于是以一维数组方式表示，所以如果要检测上下点则需要加减一行的大小。

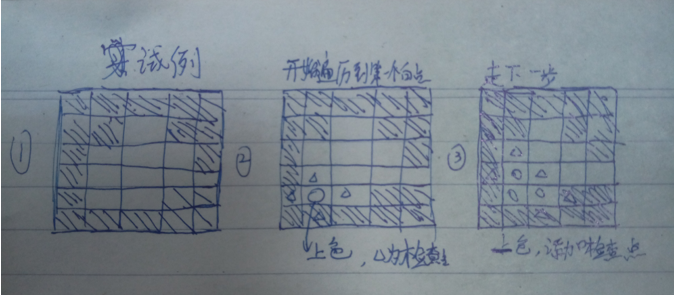
读取时指针以三个三个为单位走

**难点二**在于如何分辨不同细胞，同样都是白色，不同区域的白色要上不同的颜色。



解决方案：对于所有像素点开始遍历，定义一个变量first=0代表是否初次遇见白色，如果遇见，将first变为1，然后将初始点随机上色。对于这个已经上色的点，以他为中心，探测上下左右四个像素点，如果为白色，同样进行上色，具体算法如下：

图片展示过程：



代码展示：

每一次上色的代码：，其中number值与第二个难点有关

for (int i = 0; i < m\_nImage; i = i + 3)

{

fred = m\_pImage[i];

fgreen = m\_pImage[i + 1];

fblue = m\_pImage[i + 2];

if (fred == 255 && fgreen == 255 && fblue == 255 && first == 0)

{

m\_pImage[i] = RED;

m\_pImage[i + 1] = GREEN;

m\_pImage[i + 2] = BLUE;

first = 1;

number++;

}

if (m\_pImage[i] == RED && m\_pImage[i + 1] == GREEN && m\_pImage[i + 2] == BLUE)

{

int x1 = i + 3, x2 = i - 3, y1 = i + m\_nLineByte, y2 = i - m\_nLineByte;

if (m\_pImage[x1] == 255 && m\_pImage[x1 + 1] == 255 && m\_pImage[x1 + 2] == 255)

{

m\_pImage[x1] = RED;

m\_pImage[x1 + 1] = GREEN;

m\_pImage[x1 + 2] = BLUE;

number++;

}

if (m\_pImage[x2] == 255 && m\_pImage[x2 + 1] == 255 && m\_pImage[x2 + 2] == 255)

{

m\_pImage[x2] = RED;

m\_pImage[x2 + 1] = GREEN;

m\_pImage[x2 + 2] = BLUE;

i = i - 6;

number++;

}

if (m\_pImage[y1] == 255 && m\_pImage[y1 + 1] == 255 && m\_pImage[y1 + 2] == 255)

{

m\_pImage[y1] = RED;

m\_pImage[y1 + 1] = GREEN;

m\_pImage[y1 + 2] = BLUE;

number++;

}

if (m\_pImage[y2] == 255 && m\_pImage[y2 + 1] == 255 && m\_pImage[y2 + 2] == 255)

{

m\_pImage[y2] = RED;

m\_pImage[y2 + 1] = GREEN;

m\_pImage[y2 + 2] = BLUE;

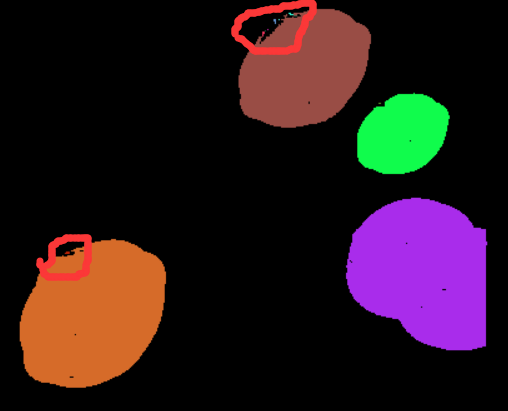
number++;

}

}

}

**难点三**在于对于最初解决难点二的方案来说，细胞图像周围可能有很多小白点，他们是不必要上色的错误点，要如何规避。



解决方案：

在给像素点上色时，记录上色数量，若小于一定数量，则说明是不用上色的小白框，将这部分已经上好的颜色替换成黑色。

代码展示：

if (number<300)

for (int i = 0; i < m\_nImage; i = i + 3)

{

fred = m\_pImage[i];

fgreen = m\_pImage[i + 1];

fblue = m\_pImage[i + 2];

if (fred == RED && fgreen == GREEN && fblue == BLUE)

{

m\_pImage[i] = 0;

m\_pImage[i + 1] = 0;

m\_pImage[i + 2] = 0;

}

}

1. 调试错误分析
2. &&ii.

第一次时不是以颜色自身为单位直接检测上下左右的像素点，而是标记上方三格添加入Vector中，且不是一次性读完全部图片而是一下一下移动指针读，直到指针本身走到最小标记点时才检测，导致上色区域为右上方三角形且如果中途碰到细胞中的黑点导致检测中断：

算法意思如图：

对于指针移动的总步数有个标记，走到2时该点为白色，添加5,6,7三个值入vector，当指针移动了5次之后，也即走到了5位置，判断该点是否为需要上色的白色点，如是，则上色并继续添加上方的三个值入vector，不是则指针前进

代码展示：

for (int i = 0; i < m\_nImage/3; i++)

{

fread(&red, sizeof(char), 1, fpo);

fread(&green, sizeof(char), 1, fpo);

fread(&blue, sizeof(char), 1, fpo);

if (green == 255 && red == 255 && blue == 255 && first == 0)

{

check.push\_back(i);

check.push\_back(i + m\_nWidth - 1);

check.push\_back(i + m\_nWidth);

check.push\_back(i + m\_nWidth + 1);

green = 0;

red = blue = 100;

first = 1;

}

if (countWidth == m\_nWidth \* 3)

{

countWidth = 0;

number++;

}

if (!check.empty())

if (i == check[m])

{

if (green == 255 && red == 255 && blue == 255)

{

int n = k;

if (check[n - 2] != (i + m\_nWidth - 1))

if ((i + m\_nWidth - 1) < (number + 1)\*m\_nWidth \* 3)

{

k++;

check.push\_back(i + m\_nWidth - 1);

}

if (check[n - 1] != (i + m\_nWidth))

if ((i + m\_nWidth)<(number + 1)\*m\_nWidth \* 3)

{

k++;

check.push\_back(i + m\_nWidth);

}

if ((i + m\_nWidth + 1) < (number + 1)\*m\_nWidth \* 3)

{

k++;

check.push\_back(i + m\_nWidth + 1);

}

green = 0;

red = blue = 100;

}

m++;

}

sort(check.begin(), check.end());

check.erase(unique(check.begin(), check.end()), check.end());

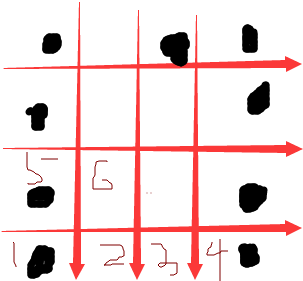
countWidth ++;

fwrite(&red, sizeof(char), 1, fpw);

fwrite(&green, sizeof(char), 1, fpw);

fwrite(&blue, sizeof(char), 1, fpw);

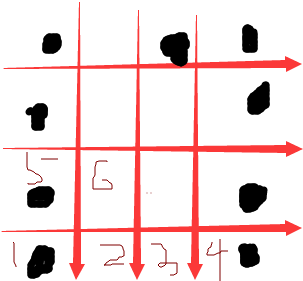
}



出现的问题如图：

手动上色上次依次如下



那么很自然的想到，原先的上方三区块添加不够，又继而向右侧也添加一个区块即添加3,5,6,7入vector，但是这样有个问题，如何保证vector添加的检查点有顺序，也即对于2他要添加 3 5 6 7入vector，进行到三后 他添加了 4 7 8 9，则vector 中顺序为 3 5 6 7 4 7 8 9，需要进入重排序与去重，则想到每次都运用STL库中的sort 与erase进行排序去重

但是这样算法复杂度极高，开始运行后一个像素点的填充进行了40多秒还没出结果，于是放弃这一个边读边上色算法

改为一次性读完，再回头检测。

正确代码在上面的自动上色中已展示

主要差别在于fread(m\_pImage, m\_nImage, 1, fpo);一次性读完

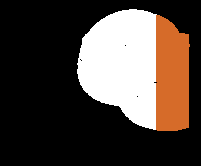
与边读边改变

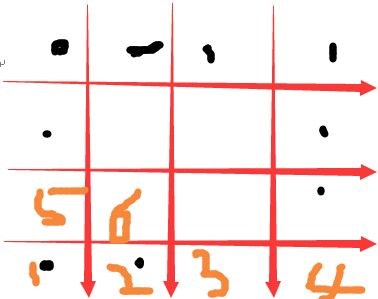
fread(&red, sizeof(char), 1, fpo);

fread(&green, sizeof(char), 1, fpo);

fread(&blue, sizeof(char), 1, fpo);

iii.在十字填充算法时发现左侧未填充，如图：





对于3，他会检测2,4,7是否是白色，检测出白色，填充，然后指针继续走，对于4检测3,8，然后走到7，检测四周，填完色，指针继续走，但是6不再检测，故出错

改正：在检测出左侧白色并填充的时候，指针回走

if (fred == 255 && fgreen == 255 && fblue == 255 && first == 0)

{

count++;

white++;

m\_pImage[i] = RED;

m\_pImage[i + 1] = GREEN;

m\_pImage[i + 2] = BLUE;

first = 1;

}

if (m\_pImage[i] == RED && m\_pImage[i + 1] == GREEN && m\_pImage[i + 2] == BLUE)

{

int x1 = i + 3, x2 = i - 3, y1 = i + m\_nLineByte, y2 = i - m\_nLineByte;

if (m\_pImage[x1] == 255 && m\_pImage[x1 + 1] == 255 && m\_pImage[x1 + 2] == 255)

{

m\_pImage[x1] = RED;

m\_pImage[x1 + 1] = GREEN;

m\_pImage[x1 + 2] = BLUE;

number++;

}

if (m\_pImage[x2] == 255 && m\_pImage[x2 + 1] == 255 && m\_pImage[x2 + 2] == 255)

{

m\_pImage[x2] = RED;

m\_pImage[x2 + 1] = GREEN;

m\_pImage[x2 + 2] = BLUE;

i = i - 6;  **//原先的算法添加这一行**

number++;

}

if (m\_pImage[y1] == 255 && m\_pImage[y1 + 1] == 255 && m\_pImage[y1 + 2] == 255)

{

m\_pImage[y1] = RED;

m\_pImage[y1 + 1] = GREEN;

m\_pImage[y1 + 2] = BLUE;

number++;

}

if (m\_pImage[y2] == 255 && m\_pImage[y2 + 1] == 255 && m\_pImage[y2 + 2] == 255)

{

m\_pImage[y2] = RED;

m\_pImage[y2 + 1] = GREEN;

m\_pImage[y2 + 2] = BLUE;

number++;

}

**iv.**

手动上色中，发现如果是上色到一，重启后再继续手动上色，有几率消除小白块时把大细胞也一起填充为黑色

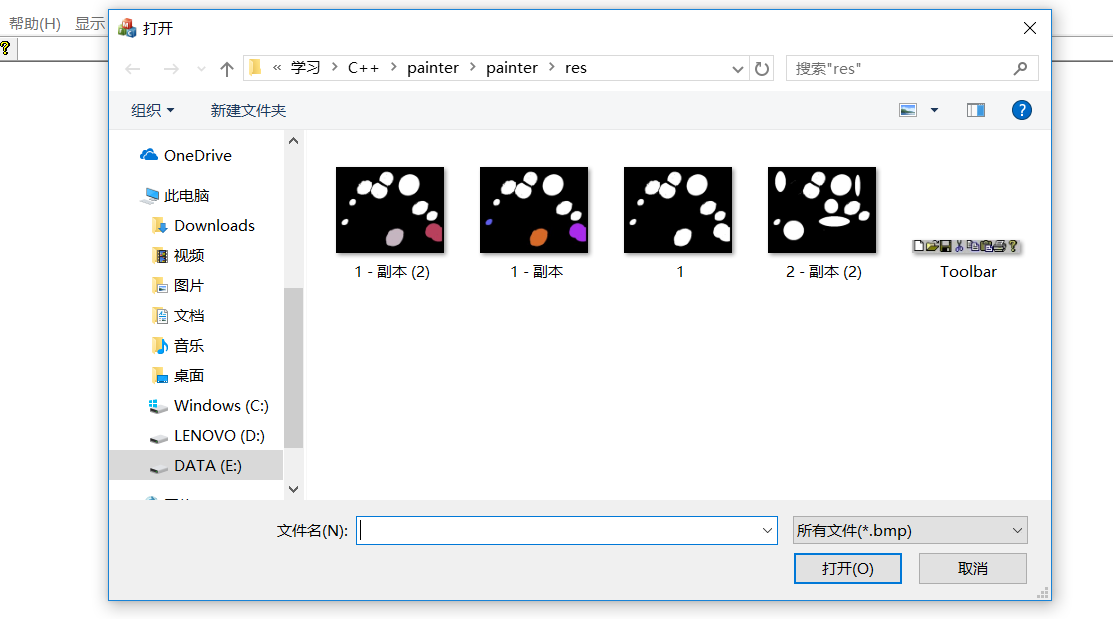
经过调试发现，使用#define Random(x) (rand() % x)，若是没有设置种子，是伪随机，特定上色步骤下，填充的是之前一样的颜色，那么如果正在填充小白块，连带会将大白快消除。

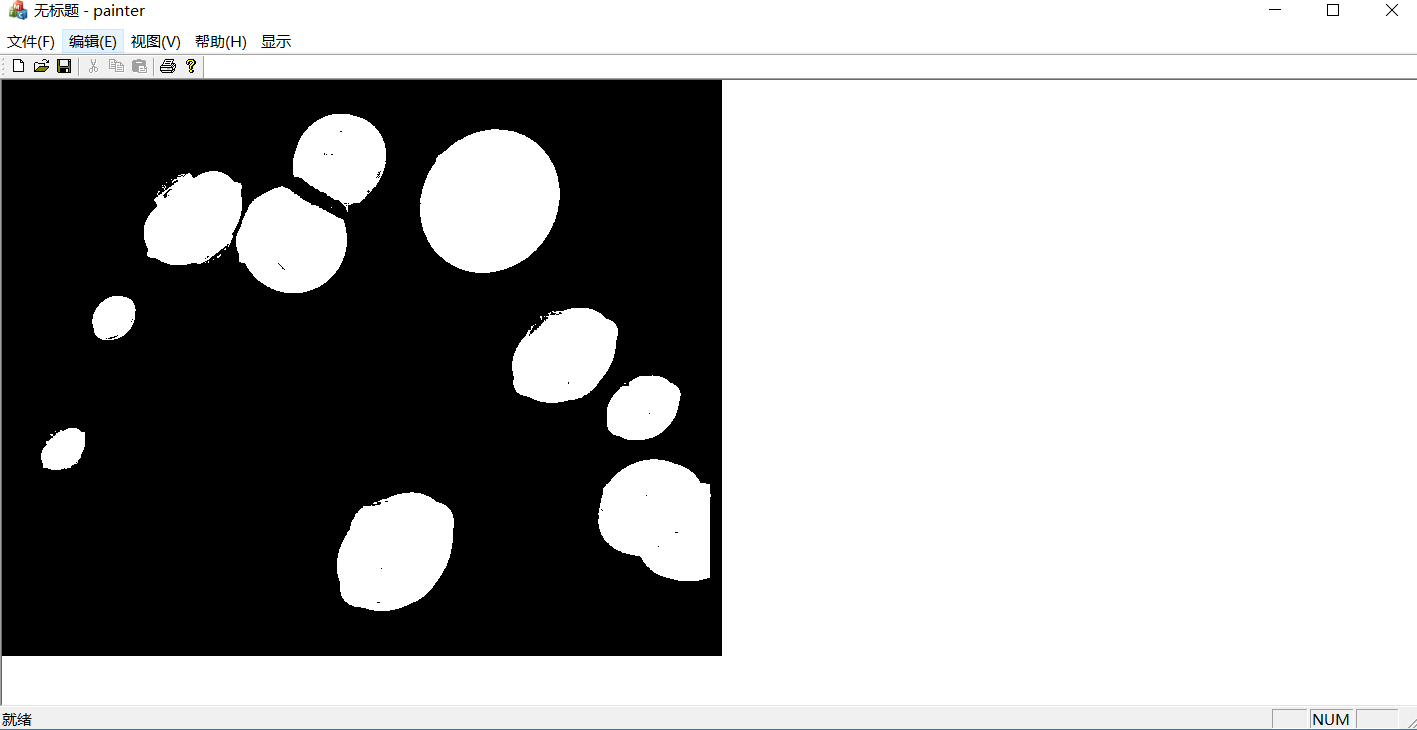
解决办法：srand((int)time(0))设定种子

**[测试结果]**

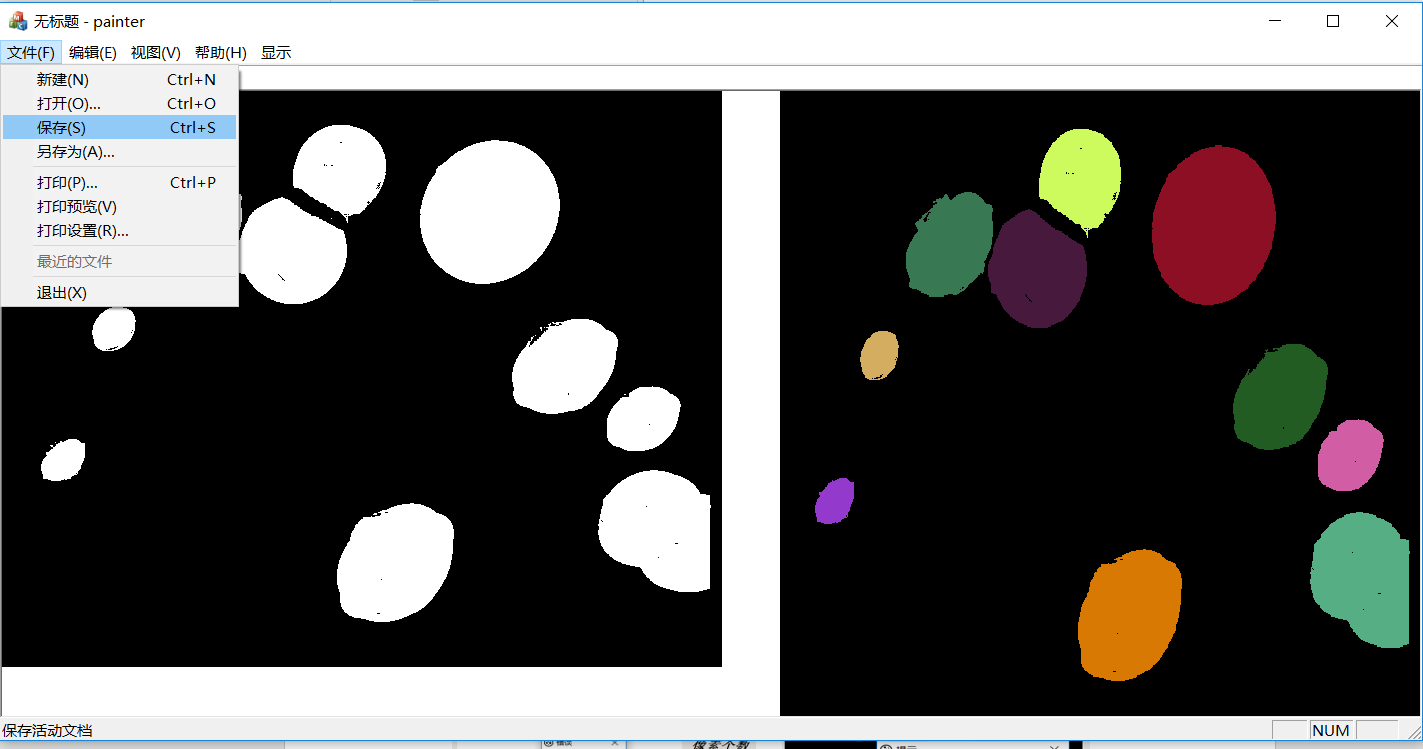
一、图像读取，保存

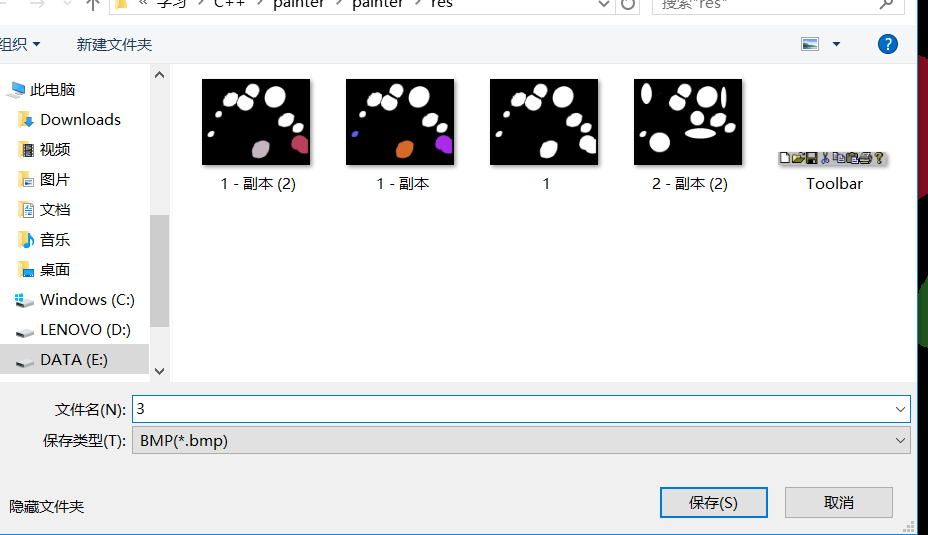
1. 打开图片

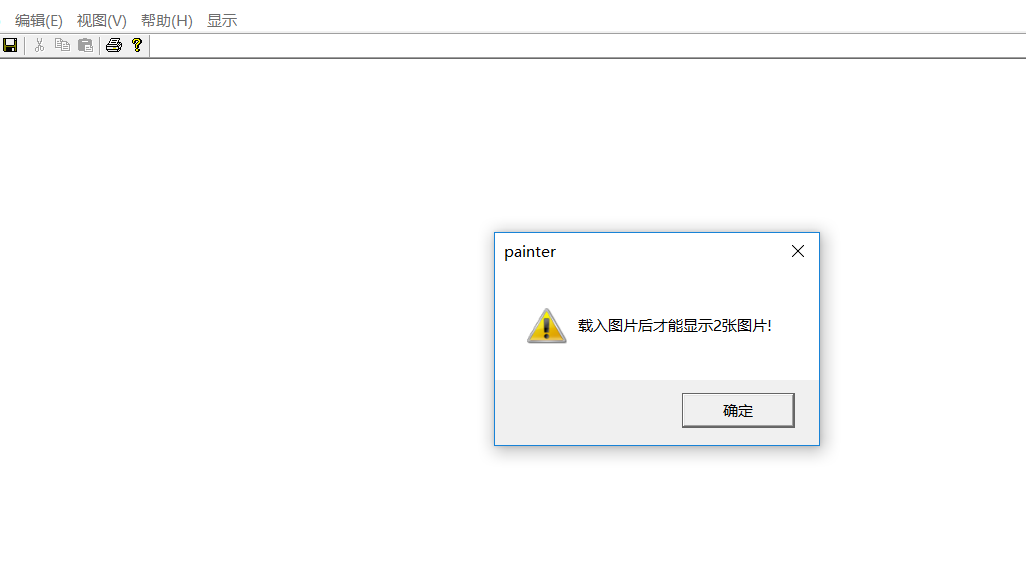


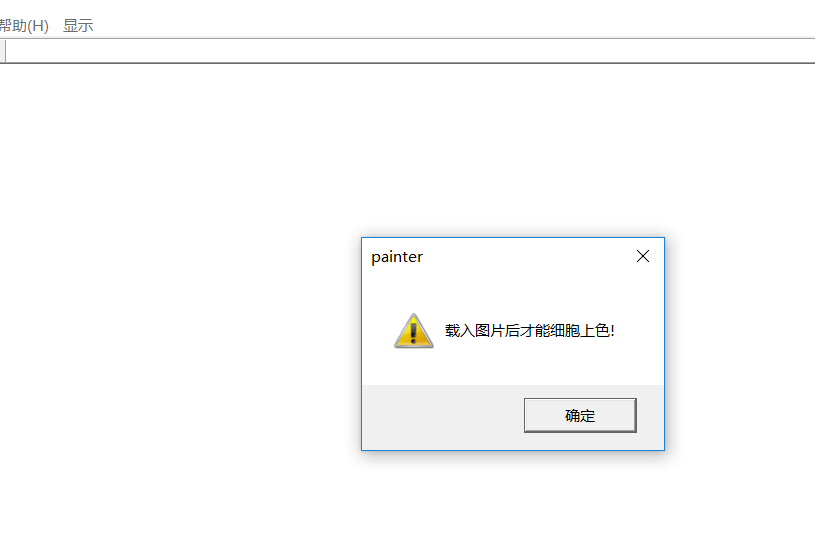


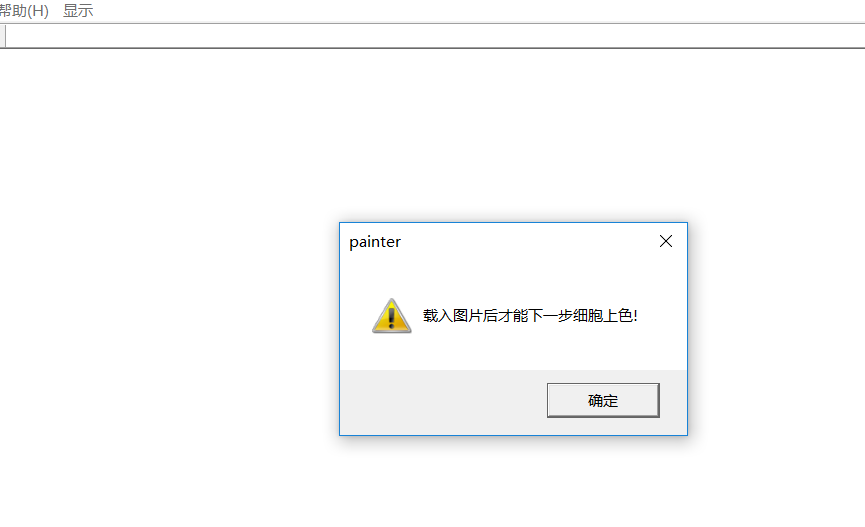
1. 保存图片





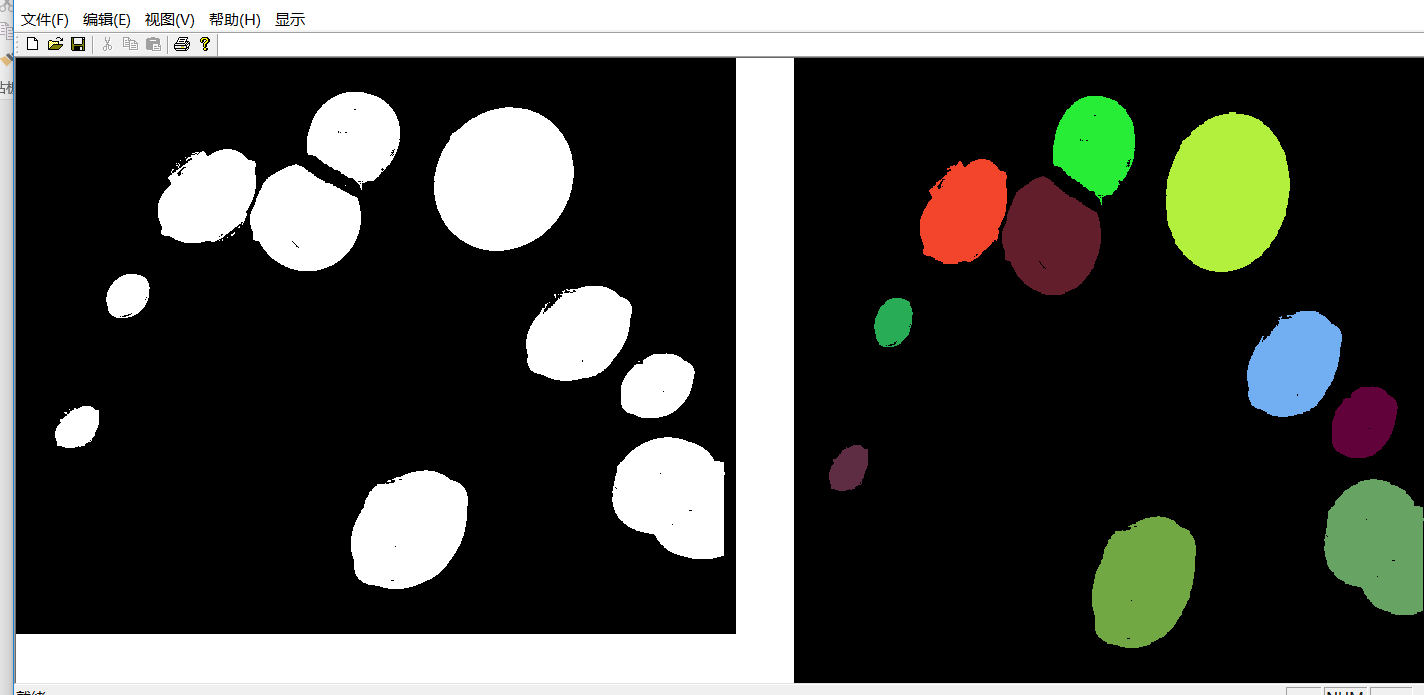
1. 异常情况处理：未打开图片就选择上色，则会出现错误提示框； 



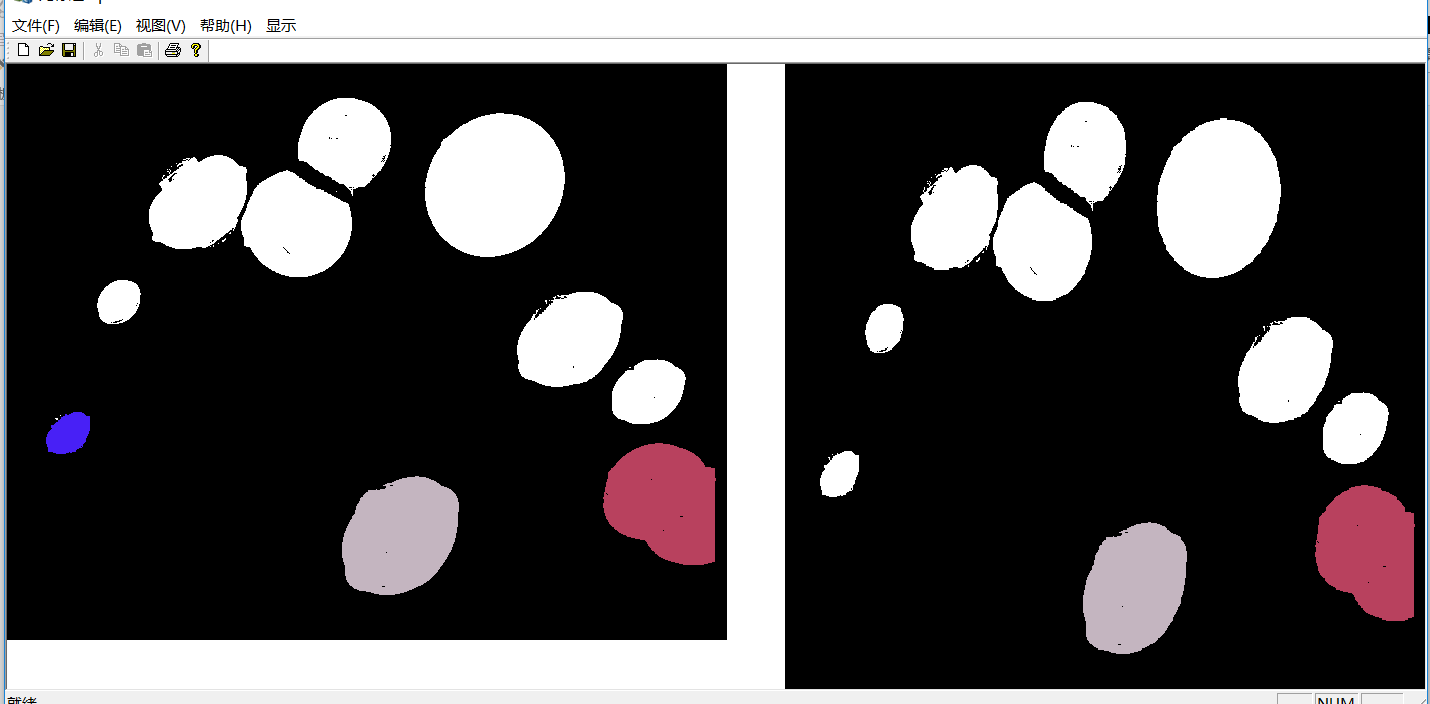


二、图像更新，细胞上色处理

1. 图像自动上色



1. 图像手动上色



**[附录]**

1. MainFrm.h

// MainFrm.h : CMainFrame 类的接口

//

#pragma once

class CMainFrame : public CFrameWnd

{

protected: // 仅从序列化创建

CMainFrame();

DECLARE\_DYNCREATE(CMainFrame)

// 特性

public:

// 操作

public:

// 重写

public:

virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// 实现

public:

virtual ~CMainFrame();

#ifdef \_DEBUG

virtual void AssertValid() const;

virtual void Dump(CDumpContext& dc) const;

#endif

protected: // 控件条嵌入成员

CToolBar m\_wndToolBar;

CStatusBar m\_wndStatusBar;

// 生成的消息映射函数

protected:

afx\_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);

DECLARE\_MESSAGE\_MAP()

};

1. painter.h

// painter.h : painter 应用程序的主头文件

//

#pragma once

#ifndef \_\_AFXWIN\_H\_\_

#error "在包含此文件之前包含“stdafx.h”以生成 PCH 文件"

#endif

#include "resource.h" // 主符号

// CpainterApp:

// 有关此类的实现，请参阅 painter.cpp

//

class CpainterApp : public CWinAppEx

{

public:

CpainterApp();

// 重写

public:

virtual BOOL InitInstance();

virtual int ExitInstance();

// 实现

afx\_msg void OnAppAbout();

DECLARE\_MESSAGE\_MAP()

};

extern CpainterApp theApp;

1. painterDoc.h

// painterDoc.h : CpainterDoc 类的接口

//

#pragma once

class CpainterDoc : public CDocument

{

protected: // 仅从序列化创建

CpainterDoc();

DECLARE\_DYNCREATE(CpainterDoc)

// 特性

public:

// 操作

public:

// 重写

public:

virtual BOOL OnNewDocument();

virtual void Serialize(CArchive& ar);

#ifdef SHARED\_HANDLERS

virtual void InitializeSearchContent();

virtual void OnDrawThumbnail(CDC& dc, LPRECT lprcBounds);

#endif // SHARED\_HANDLERS

// 实现

public:

virtual ~CpainterDoc();

#ifdef \_DEBUG

virtual void AssertValid() const;

virtual void Dump(CDumpContext& dc) const;

#endif

protected:

// 生成的消息映射函数

protected:

DECLARE\_MESSAGE\_MAP()

#ifdef SHARED\_HANDLERS

// 用于为搜索处理程序设置搜索内容的 Helper 函数

void SetSearchContent(const CString& value);

#endif // SHARED\_HANDLERS

};

1. painterView.h

// painterView.h : CpainterView 类的接口

//

#pragma once

class CpainterView : public CView

{

protected: // 仅从序列化创建

CpainterView();

DECLARE\_DYNCREATE(CpainterView)

// 特性

public:

CpainterDoc\* GetDocument() const;

public:

//添加成员函数

void ShowBitmap(CDC\* pDC, CString BmpName); //显示位图函数

bool ReadBmp(); //用来读取bmp个手机图片

bool SaveBmp(LPCSTR lpFileName); //用来保存bmp格式图片

//添加成员变量

CString EntName; //图像文件扩展名

CString BmpName; //图像文件名称

CBitmap m\_bitmap; //创建位图对象

CBitmap m\_bitmaplin; //创建临时位图对象进行处理---

CString BmpNameLin; //保存图像副本文件---

CBitmap m\_bitmaplin1; //创建临时位图对象进行处理---

CString BmpNameLin1; //保存图像副本文件---

int next = 1;

int m\_nWidth; //图像实际宽度

int m\_nHeight; //图像实际高度

int m\_nDrawWidth; //图像显示宽度

int m\_nDrawHeight; //图像显示高度

DWORD m\_nImage; //图像数据的字节数 只含位图

DWORD m\_nSize; //图像文件大小

int m\_nLineByte; //图像一行所占字节数

int m\_nBitCount; //图像每个像素所占位数

int m\_nPalette; //位图实际使用的颜色表中的颜色数

BYTE \*m\_pImage; //读入图片数据后的指针

BITMAPFILEHEADER bfh; //全局变量文件头

BITMAPINFOHEADER bih; //全局变量信息头

RGBQUAD m\_pPal; //颜色表指针

// 操作

public:

// 重写

public:

virtual void OnDraw(CDC\* pDC); // 重写以绘制该视图

virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

protected:

virtual BOOL OnPreparePrinting(CPrintInfo\* pInfo);

virtual void OnBeginPrinting(CDC\* pDC, CPrintInfo\* pInfo);

virtual void OnEndPrinting(CDC\* pDC, CPrintInfo\* pInfo);

// 实现

public:

virtual ~CpainterView();

#ifdef \_DEBUG

virtual void AssertValid() const;

virtual void Dump(CDumpContext& dc) const;

#endif

protected:

// 生成的消息映射函数

protected:

DECLARE\_MESSAGE\_MAP()

public:

afx\_msg void OnFileOpen(); //点击打开按钮

afx\_msg void OnFileSave(); //点击保存按钮

afx\_msg void OnShowTwo(); //点击显示两张图片

afx\_msg void OnNext(); //点击手动上色

afx\_msg void OnStart(); //点击自动上色

};

#ifndef \_DEBUG // painterView.cpp 中的调试版本

inline CpainterDoc\* CpainterView::GetDocument() const

{ return reinterpret\_cast<CpainterDoc\*>(m\_pDocument); }

#endif

1. MainFrm.cpp

// MainFrm.cpp : CMainFrame 类的实现

//

#include "stdafx.h"

#include "painter.h"

#include "MainFrm.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CMainFrame

IMPLEMENT\_DYNCREATE(CMainFrame, CFrameWnd)

const int iMaxUserToolbars = 10;

const UINT uiFirstUserToolBarId = AFX\_IDW\_CONTROLBAR\_FIRST + 40;

const UINT uiLastUserToolBarId = uiFirstUserToolBarId + iMaxUserToolbars - 1;

BEGIN\_MESSAGE\_MAP(CMainFrame, CFrameWnd)

ON\_WM\_CREATE()

END\_MESSAGE\_MAP()

static UINT indicators[] =

{

ID\_SEPARATOR, // 状态行指示器

ID\_INDICATOR\_CAPS,

ID\_INDICATOR\_NUM,

ID\_INDICATOR\_SCRL,

};

// CMainFrame 构造/析构

CMainFrame::CMainFrame()

{

// TODO: 在此添加成员初始化代码

}

CMainFrame::~CMainFrame()

{

}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if (CFrameWnd::OnCreate(lpCreateStruct) == -1)

return -1;

if (!m\_wndToolBar.CreateEx(this, TBSTYLE\_FLAT, WS\_CHILD | WS\_VISIBLE | CBRS\_TOP | CBRS\_GRIPPER | CBRS\_TOOLTIPS | CBRS\_FLYBY | CBRS\_SIZE\_DYNAMIC) ||

!m\_wndToolBar.LoadToolBar(IDR\_MAINFRAME))

{

TRACE0("未能创建工具栏\n");

return -1; // 未能创建

}

if (!m\_wndStatusBar.Create(this))

{

TRACE0("未能创建状态栏\n");

return -1; // 未能创建

}

m\_wndStatusBar.SetIndicators(indicators, sizeof(indicators)/sizeof(UINT));

// TODO: 如果不需要可停靠工具栏，则删除这三行

m\_wndToolBar.EnableDocking(CBRS\_ALIGN\_ANY);

EnableDocking(CBRS\_ALIGN\_ANY);

DockControlBar(&m\_wndToolBar);

return 0;

}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)

{

if( !CFrameWnd::PreCreateWindow(cs) )

return FALSE;

// TODO: 在此处通过修改

// CREATESTRUCT cs 来修改窗口类或样式

return TRUE;

}

// CMainFrame 诊断

#ifdef \_DEBUG

void CMainFrame::AssertValid() const

{

CFrameWnd::AssertValid();

}

void CMainFrame::Dump(CDumpContext& dc) const

{

CFrameWnd::Dump(dc);

}

#endif //\_DEBUG

// CMainFrame 消息处理程序

1. Painter.cpp

// painter.cpp : 定义应用程序的类行为。

//

#include "stdafx.h"

#include "afxwinappex.h"

#include "afxdialogex.h"

#include "painter.h"

#include "MainFrm.h"

#include "painterDoc.h"

#include "painterView.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CpainterApp

BEGIN\_MESSAGE\_MAP(CpainterApp, CWinAppEx)

ON\_COMMAND(ID\_APP\_ABOUT, &CpainterApp::OnAppAbout)

// 基于文件的标准文档命令

ON\_COMMAND(ID\_FILE\_NEW, &CWinAppEx::OnFileNew)

ON\_COMMAND(ID\_FILE\_OPEN, &CWinAppEx::OnFileOpen)

// 标准打印设置命令

ON\_COMMAND(ID\_FILE\_PRINT\_SETUP, &CWinAppEx::OnFilePrintSetup)

END\_MESSAGE\_MAP()

// CpainterApp 构造

CpainterApp::CpainterApp()

{

// 支持重新启动管理器

m\_dwRestartManagerSupportFlags = AFX\_RESTART\_MANAGER\_SUPPORT\_ALL\_ASPECTS;

#ifdef \_MANAGED

// 如果应用程序是利用公共语言运行时支持(/clr)构建的，则:

// 1) 必须有此附加设置，“重新启动管理器”支持才能正常工作。

// 2) 在您的项目中，您必须按照生成顺序向 System.Windows.Forms 添加引用。

System::Windows::Forms::Application::SetUnhandledExceptionMode(System::Windows::Forms::UnhandledExceptionMode::ThrowException);

#endif

// TODO: 将以下应用程序 ID 字符串替换为唯一的 ID 字符串；建议的字符串格式

//为 CompanyName.ProductName.SubProduct.VersionInformation

SetAppID(\_T("painter.AppID.NoVersion"));

// TODO: 在此处添加构造代码，

// 将所有重要的初始化放置在 InitInstance 中

}

// 唯一的一个 CpainterApp 对象

CpainterApp theApp;

// CpainterApp 初始化

BOOL CpainterApp::InitInstance()

{

// 如果一个运行在 Windows XP 上的应用程序清单指定要

// 使用 ComCtl32.dll 版本 6 或更高版本来启用可视化方式，

//则需要 InitCommonControlsEx()。 否则，将无法创建窗口。

INITCOMMONCONTROLSEX InitCtrls;

InitCtrls.dwSize = sizeof(InitCtrls);

// 将它设置为包括所有要在应用程序中使用的

// 公共控件类。

InitCtrls.dwICC = ICC\_WIN95\_CLASSES;

InitCommonControlsEx(&InitCtrls);

CWinAppEx::InitInstance();

// 初始化 OLE 库

if (!AfxOleInit())

{

AfxMessageBox(IDP\_OLE\_INIT\_FAILED);

return FALSE;

}

AfxEnableControlContainer();

EnableTaskbarInteraction(FALSE);

// 使用 RichEdit 控件需要 AfxInitRichEdit2()

// AfxInitRichEdit2();

// 标准初始化

// 如果未使用这些功能并希望减小

// 最终可执行文件的大小，则应移除下列

// 不需要的特定初始化例程

// 更改用于存储设置的注册表项

// TODO: 应适当修改该字符串，

// 例如修改为公司或组织名

SetRegistryKey(\_T("应用程序向导生成的本地应用程序"));

LoadStdProfileSettings(4); // 加载标准 INI 文件选项(包括 MRU)

// 注册应用程序的文档模板。 文档模板

// 将用作文档、框架窗口和视图之间的连接

CSingleDocTemplate\* pDocTemplate;

pDocTemplate = new CSingleDocTemplate(

IDR\_MAINFRAME,

RUNTIME\_CLASS(CpainterDoc),

RUNTIME\_CLASS(CMainFrame), // 主 SDI 框架窗口

RUNTIME\_CLASS(CpainterView));

if (!pDocTemplate)

return FALSE;

AddDocTemplate(pDocTemplate);

// 分析标准 shell 命令、DDE、打开文件操作的命令行

CCommandLineInfo cmdInfo;

ParseCommandLine(cmdInfo);

// 调度在命令行中指定的命令。 如果

// 用 /RegServer、/Register、/Unregserver 或 /Unregister 启动应用程序，则返回 FALSE。

if (!ProcessShellCommand(cmdInfo))

return FALSE;

// 唯一的一个窗口已初始化，因此显示它并对其进行更新

m\_pMainWnd->ShowWindow(SW\_SHOW);

m\_pMainWnd->UpdateWindow();

return TRUE;

}

int CpainterApp::ExitInstance()

{

//TODO: 处理可能已添加的附加资源

AfxOleTerm(FALSE);

return CWinAppEx::ExitInstance();

}

// CpainterApp 消息处理程序

// 用于应用程序“关于”菜单项的 CAboutDlg 对话框

class CAboutDlg : public CDialogEx

{

public:

CAboutDlg();

// 对话框数据

#ifdef AFX\_DESIGN\_TIME

enum { IDD = IDD\_ABOUTBOX };

#endif

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持

// 实现

protected:

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg::CAboutDlg() : CDialogEx(IDD\_ABOUTBOX)

{

}

void CAboutDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

}

BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)

END\_MESSAGE\_MAP()

// 用于运行对话框的应用程序命令

void CpainterApp::OnAppAbout()

{

CAboutDlg aboutDlg;

aboutDlg.DoModal();

}

// CpainterApp 消息处理程序

1. painterDoc.cpp

// painterDoc.cpp : CpainterDoc 类的实现

//

#include "stdafx.h"

// SHARED\_HANDLERS 可以在实现预览、缩略图和搜索筛选器句柄的

// ATL 项目中进行定义，并允许与该项目共享文档代码。

#ifndef SHARED\_HANDLERS

#include "painter.h"

#endif

#include "painterDoc.h"

#include <propkey.h>

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CpainterDoc

IMPLEMENT\_DYNCREATE(CpainterDoc, CDocument)

BEGIN\_MESSAGE\_MAP(CpainterDoc, CDocument)

END\_MESSAGE\_MAP()

// CpainterDoc 构造/析构

CpainterDoc::CpainterDoc()

{

// TODO: 在此添加一次性构造代码

}

CpainterDoc::~CpainterDoc()

{

}

BOOL CpainterDoc::OnNewDocument()

{

if (!CDocument::OnNewDocument())

return FALSE;

// TODO: 在此添加重新初始化代码

// (SDI 文档将重用该文档)

return TRUE;

}

// CpainterDoc 序列化

void CpainterDoc::Serialize(CArchive& ar)

{

if (ar.IsStoring())

{

// TODO: 在此添加存储代码

}

else

{

// TODO: 在此添加加载代码

}

}

#ifdef SHARED\_HANDLERS

// 缩略图的支持

void CpainterDoc::OnDrawThumbnail(CDC& dc, LPRECT lprcBounds)

{

// 修改此代码以绘制文档数据

dc.FillSolidRect(lprcBounds, RGB(255, 255, 255));

CString strText = \_T("TODO: implement thumbnail drawing here");

LOGFONT lf;

CFont\* pDefaultGUIFont = CFont::FromHandle((HFONT) GetStockObject(DEFAULT\_GUI\_FONT));

pDefaultGUIFont->GetLogFont(&lf);

lf.lfHeight = 36;

CFont fontDraw;

fontDraw.CreateFontIndirect(&lf);

CFont\* pOldFont = dc.SelectObject(&fontDraw);

dc.DrawText(strText, lprcBounds, DT\_CENTER | DT\_WORDBREAK);

dc.SelectObject(pOldFont);

}

// 搜索处理程序的支持

void CpainterDoc::InitializeSearchContent()

{

CString strSearchContent;

// 从文档数据设置搜索内容。

// 内容部分应由“;”分隔

// 例如: strSearchContent = \_T("point;rectangle;circle;ole object;")；

SetSearchContent(strSearchContent);

}

void CpainterDoc::SetSearchContent(const CString& value)

{

if (value.IsEmpty())

{

RemoveChunk(PKEY\_Search\_Contents.fmtid, PKEY\_Search\_Contents.pid);

}

else

{

CMFCFilterChunkValueImpl \*pChunk = NULL;

ATLTRY(pChunk = new CMFCFilterChunkValueImpl);

if (pChunk != NULL)

{

pChunk->SetTextValue(PKEY\_Search\_Contents, value, CHUNK\_TEXT);

SetChunkValue(pChunk);

}

}

}

#endif // SHARED\_HANDLERS

// CpainterDoc 诊断

#ifdef \_DEBUG

void CpainterDoc::AssertValid() const

{

CDocument::AssertValid();

}

void CpainterDoc::Dump(CDumpContext& dc) const

{

CDocument::Dump(dc);

}

#endif //\_DEBUG

// CpainterDoc 命令

1. painterView.h

// painterView.cpp : CpainterView 类的实现

//

#include "stdafx.h"

// SHARED\_HANDLERS 可以在实现预览、缩略图和搜索筛选器句柄的

// ATL 项目中进行定义，并允许与该项目共享文档代码。

#ifndef SHARED\_HANDLERS

#include "painter.h"

#endif

#include "painterDoc.h"

#include "painterView.h"

#define Random(x) (rand() % x)

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

#include<iostream>

using namespace std;

// CpainterView

IMPLEMENT\_DYNCREATE(CpainterView, CView)

BEGIN\_MESSAGE\_MAP(CpainterView, CView)

// 标准打印命令

ON\_COMMAND(ID\_FILE\_PRINT, &CView::OnFilePrint)

ON\_COMMAND(ID\_FILE\_PRINT\_DIRECT, &CView::OnFilePrint)

ON\_COMMAND(ID\_FILE\_PRINT\_PREVIEW, &CView::OnFilePrintPreview)

ON\_COMMAND(ID\_FILE\_OPEN, &CpainterView::OnFileOpen)

ON\_COMMAND(ID\_FILE\_SAVE, &CpainterView::OnFileSave)

ON\_COMMAND(ID\_SHOW\_TWO, &CpainterView::OnShowTwo)

// ON\_COMMAND(ID\_SHOW\_HD, &CpainterView::OnShowHd)

ON\_COMMAND(ID\_next, &CpainterView::OnNext)

ON\_COMMAND(ID\_start, &CpainterView::OnStart)

END\_MESSAGE\_MAP()

// CpainterView 构造/析构

CpainterView::CpainterView()

{

// TODO: 在此处添加构造代码

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* numPicture变量显示图片数量

/\* 0-提示错误或未打开图片 1-显示一张图片 2-显示两张图片和处理

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int numPicture = 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* level变量显示具体的处理操作，每个处理函数中赋值该变量

/\* 0-显示2张图片 1-显示灰度图片 3-显示图片采样

/\* 2 4 8 16 32 64-不同量化等级量化图片

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int level = 0;

void CpainterView::ShowBitmap(CDC \* pDC, CString BmpName)

{

//定义bitmap指针 调用函数LoadImage装载位图

HBITMAP m\_hBitmap;

m\_hBitmap = (HBITMAP)LoadImage(NULL, BmpName, IMAGE\_BITMAP, 0, 0,

LR\_LOADFROMFILE | LR\_DEFAULTSIZE | LR\_CREATEDIBSECTION);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 1.要装载OEM图像，则设此参数值为0 OBM\_ OEM位图 OIC\_OEM图标 OCR\_OEM光标

/\* 2.BmpName要装载图片的文件名

/\* 3.装载图像类型:

/\* IMAGE\_BITMAP-装载位图 IMAGE\_CURSOR-装载光标 IMAGE\_ICON-装载图标

/\* 4.指定图标或光标的像素宽度和长度 以像素为单位

/\* 5.加载选项:

/\* IR\_LOADFROMFILE-指明由lpszName指定文件中加载图像

/\* IR\_DEFAULTSIZE-指明使用图像默认大小

/\* LR\_CREATEDIBSECTION-当uType参数为IMAGE\_BITMAP时,创建一个DIB项

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (m\_bitmap.m\_hObject)

{

m\_bitmap.Detach(); //切断CWnd和窗口联系

}

m\_bitmap.Attach(m\_hBitmap); //将句柄HBITMAP m\_hBitmap与CBitmap m\_bitmap关联

//边界

CRect rect;

GetClientRect(&rect);

//图片显示(x,y)起始坐标

int m\_showX = 0;

int m\_showY = 0;

int m\_nWindowWidth = rect.right - rect.left; //计算客户区宽度

int m\_nWindowHeight = rect.bottom - rect.top; //计算客户区高度

//定义并创建一个内存设备环境DC

CDC dcBmp;

if (!dcBmp.CreateCompatibleDC(pDC)) //创建兼容性的DC

return;

BITMAP m\_bmp; //临时bmp图片变量

m\_bitmap.GetBitmap(&m\_bmp); //将图片载入位图中

CBitmap \*pbmpOld = NULL;

dcBmp.SelectObject(&m\_bitmap); //将位图选入临时内存设备环境

//图片显示调用函数stretchBlt

pDC->StretchBlt(0, 0, m\_bmp.bmWidth, m\_bmp.bmHeight, &dcBmp, 0, 0,

m\_bmp.bmWidth, m\_bmp.bmHeight, SRCCOPY);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* BOOL StretchBlt(int x,int y,int nWidth,int nHeight,CDC\* pSrcDC,

/\* int xSrc,int ySrc,int nSrcWidth,int nSrcHeight,DWORD dwRop );

/\* 1.参数x、y位图目标矩形左上角x、y的坐标值

/\* 2.nWidth、nHeigth位图目标矩形的逻辑宽度和高度

/\* 3.pSrcDC表示源设备CDC指针

/\* 4.xSrc、ySrc表示位图源矩形的左上角的x、y逻辑坐标值

/\* 5.dwRop表示显示位图的光栅操作方式 SRCCOPY用于直接将位图复制到目标环境中

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

dcBmp.SelectObject(pbmpOld); //恢复临时DC的位图

DeleteObject(&m\_bitmap); //删除内存中的位图

dcBmp.DeleteDC(); //删除CreateCompatibleDC得到的图片DC

/\*\*

\* 面代码为后面显示第二张图片

\*/

if (numPicture == 2) {

//显示图片函数LoadImage

HBITMAP m\_hBitmapChange=NULL;

if (level == 0) //显示2张图 BmpNameLin原图

{

m\_hBitmapChange = (HBITMAP)LoadImage(NULL, BmpName, IMAGE\_BITMAP, 0, 0,

LR\_LOADFROMFILE | LR\_DEFAULTSIZE | LR\_CREATEDIBSECTION);

}

else

if (level == 1) //灰度图片 BmpNameLin临时图片

{

m\_hBitmapChange = (HBITMAP)LoadImage(NULL, BmpNameLin, IMAGE\_BITMAP, 0, 0,

LR\_LOADFROMFILE | LR\_DEFAULTSIZE | LR\_CREATEDIBSECTION);

}

if (m\_bitmap.m\_hObject) {

m\_bitmap.Detach(); //m\_bitmap为创建的位图对象

}

m\_bitmap.Attach(m\_hBitmapChange);

//定义并创建一个内存设备环境

CDC dcBmp;

if (!dcBmp.CreateCompatibleDC(pDC)) //创建兼容性的DC

return;

BITMAP m\_bmp; //临时bmp图片变量

m\_bitmap.GetBitmap(&m\_bmp); //将图片载入位图中

CBitmap \*pbmpOld = NULL;

dcBmp.SelectObject(&m\_bitmap); //将位图选入临时内存设备环境

//如果图片太大显示大小为固定640\*640 否则显示原图大小

if (m\_nDrawWidth<650 && m\_nDrawHeight<650)

pDC->StretchBlt(m\_nWindowWidth - m\_nDrawWidth, 0,

m\_nDrawWidth, m\_nDrawHeight, &dcBmp, 0, 0, m\_bmp.bmWidth, m\_bmp.bmHeight, SRCCOPY);

else

pDC->StretchBlt(m\_nWindowWidth - 640, 0, 640, 640, &dcBmp, 0, 0,

m\_bmp.bmWidth, m\_bmp.bmHeight, SRCCOPY);

//恢复临时DC的位图

dcBmp.SelectObject(pbmpOld);

}

}

bool CpainterView::ReadBmp()

{

//图片读出存储其中的东西

FILE \*fp = fopen(BmpName, "rb");

if (fp == 0)

{

AfxMessageBox("无法打开文件!", MB\_OK, 0);

return 0;

}

//读取文件头 解决BMP格式倒置的方法

fread(&bfh.bfType, sizeof(WORD), 1, fp);

fread(&bfh.bfSize, sizeof(DWORD), 1, fp);

fread(&bfh.bfReserved1, sizeof(WORD), 1, fp);

fread(&bfh.bfReserved2, sizeof(WORD), 1, fp);

fread(&bfh.bfOffBits, sizeof(DWORD), 1, fp);

//图像文件的总字节数

m\_nSize = bfh.bfSize;

//判断是否是bmp格式图片

if (bfh.bfType != 0x4d42) //'BM'

{

AfxMessageBox("不是BMP格式图片!", MB\_OK, 0);

return 0;

}

//读取信息头

fread(&bih.biSize, sizeof(DWORD), 1, fp);

fread(&bih.biWidth, sizeof(LONG), 1, fp);

fread(&bih.biHeight, sizeof(LONG), 1, fp);

fread(&bih.biPlanes, sizeof(WORD), 1, fp);

fread(&bih.biBitCount, sizeof(WORD), 1, fp);

fread(&bih.biCompression, sizeof(DWORD), 1, fp);

fread(&bih.biSizeImage, sizeof(DWORD), 1, fp);

fread(&bih.biXPelsPerMeter, sizeof(LONG), 1, fp);

fread(&bih.biYPelsPerMeter, sizeof(LONG), 1, fp);

fread(&bih.biClrUsed, sizeof(DWORD), 1, fp);

fread(&bih.biClrImportant, sizeof(DWORD), 1, fp);

if (bih.biSize != sizeof(bih))

{

AfxMessageBox("本结构所占用字节数出现错误");

return 0;

}

//位图压缩类型，必须是 0（不压缩） 1（BI\_RLE8压缩类型）或2（BI\_RLE压缩类型）之一

if (bih.biCompression == BI\_RLE8 || bih.biCompression == BI\_RLE4)

{

AfxMessageBox("位图被压缩!");

return 0;

}

//获取图像高宽和每个像素所占位数

m\_nHeight = bih.biHeight;

m\_nWidth = bih.biWidth;

m\_nDrawHeight = bih.biHeight;

m\_nDrawWidth = bih.biWidth;

m\_nBitCount = bih.biBitCount; //每个像素所占位数

//计算图像每行像素所占的字节数（必须是32的倍数）

m\_nLineByte = (m\_nWidth\*m\_nBitCount + 31) / 32 \* 4;

//图片大小 调用系统自带的文件头 BITMAPFILEHEADER bfh; BITMAPINFOHEADER bih;

//否则用 BITMAPFILEHEADER\_ bfh; BITMAPINFOHEADER\_ bih;要 m\_nImage = m\_nLineByte \* m\_nHeight - 2;

m\_nImage = m\_nLineByte \* m\_nHeight;

//位图实际使用的颜色表中的颜色数 biClrUsed

m\_nPalette = 0; //初始化

if (bih.biClrUsed)

m\_nPalette = bih.biClrUsed;

//申请位图空间 大小为位图大小 m\_nImage

//malloc只能申请4字节的空间 （未知）

m\_pImage = (BYTE\*)malloc(m\_nImage);

fread(m\_pImage, m\_nImage, 1, fp);

fclose(fp);

return true;

}

bool CpainterView::SaveBmp(LPCSTR lpFileName)

{

//保存bmp格式图片 写图片过程 只处理24像素的图片 该图片无调色板

FILE \*fpo = fopen(BmpNameLin, "rb");

FILE \*fpw = fopen(lpFileName, "wb");

fread(&bfh, sizeof(BITMAPFILEHEADER), 1, fpo);

fread(&bih, sizeof(BITMAPINFOHEADER), 1, fpo);

fwrite(&bfh, sizeof(BITMAPFILEHEADER), 1, fpw);

fwrite(&bih, sizeof(BITMAPINFOHEADER), 1, fpw);

//malloc只能申请4字节的空间 （未知）

m\_pImage = (BYTE\*)malloc(m\_nImage);

fread(m\_pImage, m\_nImage, 1, fpo);

fwrite(m\_pImage, m\_nImage, 1, fpw);

fclose(fpo);

fclose(fpw);

return true;

}

CpainterView::~CpainterView()

{

}

BOOL CpainterView::PreCreateWindow(CREATESTRUCT& cs)

{

// TODO: 在此处通过修改

// CREATESTRUCT cs 来修改窗口类或样式

return CView::PreCreateWindow(cs);

}

// CpainterView 绘制

void CpainterView::OnDraw(CDC\* pDC)

{

CpainterDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

if (!pDoc)

return;

if (EntName.Compare("bmp") == 0) //bmp格式

{

ReadBmp();

ShowBitmap(pDC, BmpName); //显示图片

}

// TODO: 在此处为本机数据添加绘制代码

}

// CpainterView 打印

BOOL CpainterView::OnPreparePrinting(CPrintInfo\* pInfo)

{

// 默认准备

return DoPreparePrinting(pInfo);

}

void CpainterView::OnBeginPrinting(CDC\* /\*pDC\*/, CPrintInfo\* /\*pInfo\*/)

{

// TODO: 添加额外的打印前进行的初始化过程

}

void CpainterView::OnEndPrinting(CDC\* /\*pDC\*/, CPrintInfo\* /\*pInfo\*/)

{

// TODO: 添加打印后进行的清理过程

}

// CpainterView 诊断

#ifdef \_DEBUG

void CpainterView::AssertValid() const

{

CView::AssertValid();

}

void CpainterView::Dump(CDumpContext& dc) const

{

CView::Dump(dc);

}

CpainterDoc\* CpainterView::GetDocument() const // 非调试版本是内联的

{

ASSERT(m\_pDocument->IsKindOf(RUNTIME\_CLASS(CpainterDoc)));

return (CpainterDoc\*)m\_pDocument;

}

#endif //\_DEBUG

// CpainterView 消息处理程序

void CpainterView::OnFileOpen()

{

//两种格式的文件：bmp gif

CString filter;

filter = "所有文件(\*.bmp)|\*.bmp;| BMP(\*.bmp)|\*.bmp||";

CFileDialog dlg(TRUE, NULL, NULL, OFN\_HIDEREADONLY, filter, NULL);

//按下确定按钮 dlg.DoModal() 函数显示对话框

if (dlg.DoModal() == IDOK)

{

BmpName = dlg.GetPathName(); //获取文件路径名 如D:\pic\abc.bmp

BmpNameLin = "picture.bmp"; //临时变量名-----

numPicture = 1; //显示一张图片 ------

EntName = dlg.GetFileExt(); //获取文件扩展名

EntName.MakeLower(); //将文件扩展名转换为一个小写字符

Invalidate(); //调用该函数就会调用OnDraw重绘画图

}

}

void CpainterView::OnFileSave()

{

// TODO: Add your command handler code here

CString filter;

filter = "所有文件(\*.bmp)|\*.bmp;| BMP(\*.bmp)|\*.bmp||";

//重点: 1-文件打开 0-文件保存

CFileDialog dlg(0, NULL, NULL, OFN\_HIDEREADONLY, filter, NULL);

//按下确定按钮

if (dlg.DoModal() == IDOK) {

CString str;

CString strName;

CString filename;

str = dlg.GetPathName(); //获取文件的路径

filename = dlg.GetFileTitle(); //获取文件名

int nFilterIndex = dlg.m\_ofn.nFilterIndex;

if (nFilterIndex == 2) //当用户选择文件过滤器为".BMP"时

{

str = str + ".bmp"; //自动加扩展名.bmp

SaveBmp(str); //保存bmp图片 就是一个写出图片的过程

AfxMessageBox("图片保存成功", MB\_OK, 0);

}

}

}

void CpainterView::OnShowTwo()

{

//如果没有导入图片直接点击双显 提示错误

if (numPicture == 0)

{

AfxMessageBox("载入图片后才能显示2张图片!");

return;

}

AfxMessageBox("显示两张图片!", MB\_OK, 0);

numPicture = 2; //全局变量 显示两图

level = 0; //level=0双显

Invalidate(); //调用Invalidate 每秒调用一次OnDraw画图

}

void CpainterView::OnNext()

{

if (numPicture == 0)

{

AfxMessageBox("载入图片后才能下一步细胞上色!", MB\_OK, 0);

return;

}

AfxMessageBox("细胞上色!", MB\_OK, 0);

//打开临时的图片

if (next == 0)

{

FILE \*fpo = fopen(BmpNameLin, "rb");

FILE \*fpw = fopen(BmpName, "wb+");

fread(&bfh, sizeof(BITMAPFILEHEADER), 1, fpo);

fread(&bih, sizeof(BITMAPINFOHEADER), 1, fpo);

fwrite(&bfh, sizeof(BITMAPFILEHEADER), 1, fpw);

fwrite(&bih, sizeof(BITMAPINFOHEADER), 1, fpw);

fread(m\_pImage, m\_nImage, 1, fpo);

next++;

unsigned char fred, fgreen, fblue;

int number = 0;

int first = 0;

int RED = Random(255), GREEN = Random(255), BLUE = Random(255);

for (int i = 0; i < m\_nImage; i = i + 3)

{

fred = m\_pImage[i];

fgreen = m\_pImage[i + 1];

fblue = m\_pImage[i + 2];

if (fred == 255 && fgreen == 255 && fblue == 255 && first == 0)

{

m\_pImage[i] = RED;

m\_pImage[i + 1] = GREEN;

m\_pImage[i + 2] = BLUE;

first = 1;

number++;

}

if (m\_pImage[i] == RED && m\_pImage[i + 1] == GREEN && m\_pImage[i + 2] == BLUE)

{

int x1 = i + 3, x2 = i - 3, y1 = i + m\_nLineByte, y2 = i - m\_nLineByte;

if (m\_pImage[x1] == 255 && m\_pImage[x1 + 1] == 255 && m\_pImage[x1 + 2] == 255)

{

m\_pImage[x1] = RED;

m\_pImage[x1 + 1] = GREEN;

m\_pImage[x1 + 2] = BLUE;

number++;

}

if (m\_pImage[x2] == 255 && m\_pImage[x2 + 1] == 255 && m\_pImage[x2 + 2] == 255)

{

m\_pImage[x2] = RED;

m\_pImage[x2 + 1] = GREEN;

m\_pImage[x2 + 2] = BLUE;

i = i - 6;

number++;

}

if (m\_pImage[y1] == 255 && m\_pImage[y1 + 1] == 255 && m\_pImage[y1 + 2] == 255)

{

m\_pImage[y1] = RED;

m\_pImage[y1 + 1] = GREEN;

m\_pImage[y1 + 2] = BLUE;

number++;

}

if (m\_pImage[y2] == 255 && m\_pImage[y2 + 1] == 255 && m\_pImage[y2 + 2] == 255)

{

m\_pImage[y2] = RED;

m\_pImage[y2 + 1] = GREEN;

m\_pImage[y2 + 2] = BLUE;

number++;

}

}

}

if (number<300)

for (int i = 0; i < m\_nImage; i = i + 3)

{

fred = m\_pImage[i];

fgreen = m\_pImage[i + 1];

fblue = m\_pImage[i + 2];

if (fred == RED && fgreen == GREEN && fblue == BLUE)

{

m\_pImage[i] = 0;

m\_pImage[i + 1] = 0;

m\_pImage[i + 2] = 0;

}

}

fwrite(m\_pImage, m\_nImage, 1, fpw);

fclose(fpo);

fclose(fpw);

numPicture = 2;

level = 1;

Invalidate();

}

else if(next==1)

{

FILE \*fpo = fopen(BmpName, "rb");

FILE \*fpw = fopen(BmpNameLin, "wb+");

fread(&bfh, sizeof(BITMAPFILEHEADER), 1, fpo);

fread(&bih, sizeof(BITMAPINFOHEADER), 1, fpo);

fwrite(&bfh, sizeof(BITMAPFILEHEADER), 1, fpw);

fwrite(&bih, sizeof(BITMAPINFOHEADER), 1, fpw);

fread(m\_pImage, m\_nImage, 1, fpo);

next--;

unsigned char fred, fgreen, fblue;

int number = 0;

int first = 0;

int RED = Random(255), GREEN = Random(255), BLUE = Random(255);

for (int i = 0; i < m\_nImage; i = i + 3)

{

fred = m\_pImage[i];

fgreen = m\_pImage[i + 1];

fblue = m\_pImage[i + 2];

if (fred == 255 && fgreen == 255 && fblue == 255 && first == 0)

{

m\_pImage[i] = RED;

m\_pImage[i + 1] = GREEN;

m\_pImage[i + 2] = BLUE;

first = 1;

number++;

}

if (m\_pImage[i] == RED && m\_pImage[i + 1] == GREEN && m\_pImage[i + 2] == BLUE)

{

int x1 = i + 3, x2 = i - 3, y1 = i + m\_nLineByte, y2 = i - m\_nLineByte;

if (m\_pImage[x1] == 255 && m\_pImage[x1 + 1] == 255 && m\_pImage[x1 + 2] == 255)

{

m\_pImage[x1] = RED;

m\_pImage[x1 + 1] = GREEN;

m\_pImage[x1 + 2] = BLUE;

number++;

}

if (m\_pImage[x2] == 255 && m\_pImage[x2 + 1] == 255 && m\_pImage[x2 + 2] == 255)

{

m\_pImage[x2] = RED;

m\_pImage[x2 + 1] = GREEN;

m\_pImage[x2 + 2] = BLUE;

i = i - 6;

number++;

}

if (m\_pImage[y1] == 255 && m\_pImage[y1 + 1] == 255 && m\_pImage[y1 + 2] == 255)

{

m\_pImage[y1] = RED;

m\_pImage[y1 + 1] = GREEN;

m\_pImage[y1 + 2] = BLUE;

number++;

}

if (m\_pImage[y2] == 255 && m\_pImage[y2 + 1] == 255 && m\_pImage[y2 + 2] == 255)

{

m\_pImage[y2] = RED;

m\_pImage[y2 + 1] = GREEN;

m\_pImage[y2 + 2] = BLUE;

number++;

}

}

}

if (number<300)

for (int i = 0; i < m\_nImage; i = i + 3)

{

fred = m\_pImage[i];

fgreen = m\_pImage[i + 1];

fblue = m\_pImage[i + 2];

if (fred == RED && fgreen == GREEN && fblue == BLUE)

{

m\_pImage[i] = 0;

m\_pImage[i + 1] = 0;

m\_pImage[i + 2] = 0;

}

}

fwrite(m\_pImage, m\_nImage, 1, fpw);

fclose(fpo);

fclose(fpw);

numPicture = 2;

level = 1;

Invalidate();

}

}

void CpainterView::OnStart()

{

if (numPicture == 0)

{

AfxMessageBox("载入图片后才能细胞上色!", MB\_OK, 0);

return;

}

AfxMessageBox("细胞上色!", MB\_OK, 0);

//打开临时的图片

FILE \*fpo = fopen(BmpName, "rb");

FILE \*fpw = fopen(BmpNameLin, "wb+");

//读取文件

fread(&bfh, sizeof(BITMAPFILEHEADER), 1, fpo);

fread(&bih, sizeof(BITMAPINFOHEADER), 1, fpo);

fwrite(&bfh, sizeof(BITMAPFILEHEADER), 1, fpw);

fwrite(&bih, sizeof(BITMAPINFOHEADER), 1, fpw);

unsigned char fred, fgreen, fblue;

fread(m\_pImage, m\_nImage, 1, fpo);

int white=1;

int count=0;

for(white;white>0;white--)

{

int first = 0;

int number = 0;

srand((int)time(0));

int RED = (rand() % 255), GREEN = (rand() % 255), BLUE = (rand() % 255);

for(int k=count;k>0;k--)

RED = (rand() % 255), GREEN = (rand() % 255), BLUE = (rand() % 255);

for (int i = 0; i < m\_nImage; i = i + 3)

{

fred = m\_pImage[i];

fgreen = m\_pImage[i + 1];

fblue = m\_pImage[i + 2];

if (fred == 255 && fgreen == 255 && fblue == 255 && first == 0)

{

count++;

white++;

m\_pImage[i] = RED;

m\_pImage[i + 1] = GREEN;

m\_pImage[i + 2] = BLUE;

first = 1;

}

if (m\_pImage[i] == RED && m\_pImage[i + 1] == GREEN && m\_pImage[i + 2] == BLUE)

{

int x1 = i + 3, x2 = i - 3, y1 = i + m\_nLineByte, y2 = i - m\_nLineByte;

if (m\_pImage[x1] == 255 && m\_pImage[x1 + 1] == 255 && m\_pImage[x1 + 2] == 255)

{

m\_pImage[x1] = RED;

m\_pImage[x1 + 1] = GREEN;

m\_pImage[x1 + 2] = BLUE;

number++;

}

if (m\_pImage[x2] == 255 && m\_pImage[x2 + 1] == 255 && m\_pImage[x2 + 2] == 255)

{

m\_pImage[x2] = RED;

m\_pImage[x2 + 1] = GREEN;

m\_pImage[x2 + 2] = BLUE;

i = i - 6;

number++;

}

if (m\_pImage[y1] == 255 && m\_pImage[y1 + 1] == 255 && m\_pImage[y1 + 2] == 255)

{

m\_pImage[y1] = RED;

m\_pImage[y1 + 1] = GREEN;

m\_pImage[y1 + 2] = BLUE;

number++;

}

if (m\_pImage[y2] == 255 && m\_pImage[y2 + 1] == 255 && m\_pImage[y2 + 2] == 255)

{

m\_pImage[y2] = RED;

m\_pImage[y2 + 1] = GREEN;

m\_pImage[y2 + 2] = BLUE;

number++;

}

}

}

if (number<300)

for (int i = 0; i < m\_nImage; i = i + 3)

{

fred = m\_pImage[i];

fgreen = m\_pImage[i + 1];

fblue = m\_pImage[i + 2];

if (fred == RED && fgreen == GREEN && fblue == BLUE)

{

m\_pImage[i] = 0;

m\_pImage[i + 1] = 0;

m\_pImage[i + 2] = 0;

}

}

}

fwrite(m\_pImage, m\_nImage, 1, fpw);

fclose(fpo);

fclose(fpw);

numPicture = 2;

level = 1;

Invalidate();

}