// MFC测试Dlg.cpp : 实现文件

#include "stdafx.h"

#include "MFC测试.h"

#include "MFC测试Dlg.h"

#include "afxdialogex.h"

#include<opencv2/opencv.hpp>

#include <iostream>

using namespace cv;

using namespace std;

//-----------------------------------【全局变量声明部分】--------------------------------------

// 描述：全局变量的声明

//-----------------------------------------------------------------------------------------------

CString strFilePath;

CString strFileName;

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// 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

enum { IDD = IDD\_ABOUTBOX };

protected:

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

// 实现

protected:

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)

{

}

void CAboutDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

}

BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)

END\_MESSAGE\_MAP()

// CMFC测试Dlg 对话框

CMFC测试Dlg::CMFC测试Dlg(CWnd\* pParent /\*=NULL\*/)

: CDialogEx(CMFC测试Dlg::IDD, pParent)

{

m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

}

void CMFC测试Dlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

}

BEGIN\_MESSAGE\_MAP(CMFC测试Dlg, CDialogEx)

ON\_WM\_SYSCOMMAND()

ON\_WM\_PAINT()

ON\_WM\_QUERYDRAGICON()

ON\_BN\_CLICKED(IDC\_BUTTON1, &CMFC测试Dlg::OnBnClickedButton1)

ON\_BN\_CLICKED(IDC\_BUTTON2, &CMFC测试Dlg::OnBnClickedButton2)

ON\_BN\_CLICKED(IDC\_BUTTON3, &CMFC测试Dlg::OnBnClickedButton3)

ON\_BN\_CLICKED(IDC\_BUTTON4, &CMFC测试Dlg::OnBnClickedButton4)

ON\_BN\_CLICKED(IDC\_BUTTON5, &CMFC测试Dlg::OnBnClickedButton5)

ON\_BN\_CLICKED(IDC\_BUTTON6, &CMFC测试Dlg::OnBnClickedButton6)

ON\_BN\_CLICKED(IDC\_BUTTON7, &CMFC测试Dlg::OnBnClickedButton7)

ON\_BN\_CLICKED(IDC\_BUTTON8, &CMFC测试Dlg::OnBnClickedButton8)

ON\_EN\_CHANGE(IDC\_InputChar, &CMFC测试Dlg::OnEnChangeInputchar)

END\_MESSAGE\_MAP()

// CMFC测试Dlg 消息处理程序

BOOL CMFC测试Dlg::OnInitDialog()

{

CDialogEx::OnInitDialog();

// 将“关于...”菜单项添加到系统菜单中。

// IDM\_ABOUTBOX 必须在系统命令范围内。

ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);

ASSERT(IDM\_ABOUTBOX < 0xF000);

CMenu\* pSysMenu = GetSystemMenu(FALSE);

if (pSysMenu != NULL)

{

BOOL bNameValid;

CString strAboutMenu;

bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);

ASSERT(bNameValid);

if (!strAboutMenu.IsEmpty())

{

pSysMenu->AppendMenu(MF\_SEPARATOR);

pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);

}

}

// 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动

// 执行此操作

SetIcon(m\_hIcon, TRUE); // 设置大图标

SetIcon(m\_hIcon, FALSE); // 设置小图标

ShowWindow(SW\_MAXIMIZE);

// TODO: 在此添加额外的初始化代码

return TRUE; // 除非将焦点设置到控件，否则返回 TRUE

}

void CMFC测试Dlg::OnSysCommand(UINT nID, LPARAM lParam)

{

if ((nID & 0xFFF0) == IDM\_ABOUTBOX)

{

CAboutDlg dlgAbout;

dlgAbout.DoModal();

}

else

{

CDialogEx::OnSysCommand(nID, lParam);

}

}

// 如果向对话框添加最小化按钮，则需要下面的代码

// 来绘制该图标。对于使用文档/视图模型的 MFC 应用程序，

// 这将由框架自动完成。

void CMFC测试Dlg::OnPaint()

{

if (IsIconic())

{

CPaintDC dc(this); // 用于绘制的设备上下文

SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);

// 使图标在工作区矩形中居中

int cxIcon = GetSystemMetrics(SM\_CXICON);

int cyIcon = GetSystemMetrics(SM\_CYICON);

CRect rect;

GetClientRect(&rect);

int x = (rect.Width() - cxIcon + 1) / 2;

int y = (rect.Height() - cyIcon + 1) / 2;

// 绘制图标

dc.DrawIcon(x, y, m\_hIcon);

}

else

{

CDialogEx::OnPaint();

}

}

//当用户拖动最小化窗口时系统调用此函数取得光标

//显示。

HCURSOR CMFC测试Dlg::OnQueryDragIcon()

{

return static\_cast<HCURSOR>(m\_hIcon);

}

//轮廓程序

void lunkuo()

{

Mat g\_srcImage;

Mat g\_srcImage1;

Mat g\_grayImage;

Mat g\_grayImage1;

Mat g\_grayImage2;

Mat g\_cannyMat\_output;

vector<vector<Point>> g\_vContours;

vector<Vec4i> g\_vHierarchy;

// 加载源图像

g\_srcImage = imread( "Mosaic\_map/output\_caijian.bmp", 1 );

g\_srcImage1 = imread( "Mosaic\_map/output.bmp", 1 );

// 转成灰度并模糊化降噪

cvtColor( g\_srcImage, g\_grayImage, COLOR\_BGR2GRAY );

// blur( g\_grayImage, g\_grayImage, Size(3,3) );

GaussianBlur( g\_grayImage, g\_grayImage, Size(9,9) ,0,0);

Mat drawing = Mat::zeros(g\_grayImage.rows, g\_grayImage.cols, CV\_8UC3);

g\_grayImage1 = g\_grayImage > 80;

g\_grayImage2 = g\_grayImage > 60;

g\_grayImage1=~g\_grayImage1;

g\_grayImage2=~g\_grayImage2;

g\_grayImage=g\_grayImage1-g\_grayImage2;

g\_grayImage=g\_grayImage1-g\_grayImage;

//g\_grayImage=~g\_grayImage;

Mat element=getStructuringElement(MORPH\_RECT,Size(3,3));

//Mat element1=getStructuringElement(MORPH\_RECT,Size(15,15));

//morphologyEx(g\_grayImage,g\_grayImage,MORPH\_GRADIENT,element);

//morphologyEx(g\_grayImage,g\_grayImage,MORPH\_GRADIENT,element);

morphologyEx(g\_grayImage,g\_grayImage,MORPH\_DILATE,element);

// morphologyEx(g\_grayImage,g\_grayImage,MORPH\_GRADIENT,element1);

//https://blog.csdn.net/weixin\_35738542/article/details/52198640

//imshow("1",g\_grayImage1);

//imshow("2",g\_grayImage2);

//imshow("3",g\_grayImage);

// g\_grayImage=~g\_grayImage;

//imshow("la",g\_grayImage);

// 寻找轮廓

findContours( g\_grayImage, g\_vContours, g\_vHierarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE, Point(0, 0) );

// 绘出轮廓

for( int i = 0; i< g\_vContours.size(); i++ )

{

Scalar color = Scalar(255, 255, 0);//青色

drawContours( drawing, g\_vContours, i, color, 4, 8, g\_vHierarchy, 0, Point() );

}

//imshow("da",drawing);

// 显示效果图

resize(drawing,drawing,Size(),1,1);

Mat imageROI;

// imageROI=drawing(Range(350,0,g\_srcImage1.cols,g\_srcImage1.rows));

imageROI = g\_srcImage1(Rect(350, 0, drawing.cols, drawing.rows));

addWeighted(imageROI,0.5,drawing,0.5,0.0,imageROI);

imwrite("Defect\_map/lunkuo.bmp",g\_srcImage1);

waitKey(0);

}

//1调用裁剪过的图片，加快处理速度；

//预处理，灰度，均值滤波，减少图像的噪点和失真

//滤波处理主要是提取图像特征或者消除数字化时混入的噪声

//二值化处理，依赖于具体问题，物体在不同的图像中有可能有不同的灰度值

//0为黑色，255为白色

//1.开运算

//先腐蚀后膨胀

//作用：放大裂缝和低密度区域，消除小物体，在平滑较大物体的边界时，不改变其面积

//2.闭运算

//先膨胀后腐蚀

//作用：排除小型黑洞，突触了比原图轮廓区域更暗的区域

//掉角程序

void diaojiao()

{

//局部变量的定义

Mat g\_srcImage;

Mat g\_srcImage1;

Mat g\_grayImage;

Mat threshold\_output;

vector<vector<Point>> contours;

vector<Vec4i> hierarchy;

//调用裁剪过的图片

g\_srcImage = imread( "Mosaic\_map/output\_caijian.bmp", 1 );

//调用最终的图片

g\_srcImage1 = imread( "Mosaic\_map/output.bmp", 1 );

//对图像进行灰度处理

cvtColor( g\_srcImage, g\_grayImage, COLOR\_BGR2GRAY );

//imshow("ls",g\_grayImage);

//使用均值滤波对图像进行降噪blur

blur( g\_grayImage, g\_grayImage, Size(7,7) );

// 使用Threshold检测边缘

threshold( g\_grayImage, threshold\_output,40 , 255, THRESH\_BINARY );

//imshow("la",threshold\_output);

// 0：矩形 1：十字交叉形 2： 椭圆

Mat element=getStructuringElement(MORPH\_RECT,Size(15,15));

//开运算

morphologyEx(threshold\_output,threshold\_output,MORPH\_OPEN,element);

//闭运算

morphologyEx(threshold\_output,threshold\_output,MORPH\_CLOSE,element);

//imshow("la",threshold\_output);

//求反

threshold\_output=~threshold\_output;

//imshow("la",threshold\_output);//程序检测用，可作为显示

findContours( threshold\_output, contours, hierarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE, Point(0, 0) );// 找出轮廓

//局部变量定义——向量变量

vector<vector<Point> > contours\_poly( contours.size() );

vector<Rect> boundRect( contours.size() );

vector<Point2f>center( contours.size() );

vector<float>radius( contours.size() );

// 多边形逼近轮廓 + 获取矩形和圆形边界框

//一个循环，遍历所有部分

for( unsigned int i = 0; i < contours.size(); i++ )

{

//用指定精度逼近多边形曲线

approxPolyDP( Mat(contours[i]), contours\_poly[i], 3, true );

//计算点集的最外面（up-right）矩形边界

boundRect[i] = boundingRect( Mat(contours\_poly[i]) );

//对给定的 2D点集，寻找最小面积的包围圆形

minEnclosingCircle( contours\_poly[i], center[i], radius[i] );

}

// 绘制

Mat drawing = Mat::zeros( threshold\_output.size(), CV\_8UC3 );

for( int unsigned i = 0; i<contours.size( ); i++ )

{

//设置为黄颜色

Scalar color = Scalar(0 ,255,255);

//绘制轮廓

drawContours( drawing, contours\_poly, i, color, 4, 8, vector<Vec4i>(), 0, Point() );

//circle( drawing, center[i], (int)radius[i], color, 4, 8, 0 );//绘制圆

//计算面积与周长

//计算面积

double area = contourArea(contours[i]);

//计算周长

double length = arcLength(contours[i], true);

//计算

//面积计算

area=area\*(30.0/40.0\*1.4)/100;

//周长计算

length=length\*(30.0/40.0\*1.4)/10;

//创建文件

ofstream outfile;

//创建名为diaojiao\_canshu.txt文件，存储数据

outfile.open("diaojiao\_canshu.txt");

//数据流的显示

outfile<<"面积为"<<area<<"mm2"<<endl;

//数据流的显示

outfile<<"周长为"<<length<<"mm"<<endl;

outfile.close();//关闭文本

}

//叠加，将检测出的缺陷痕迹与裁剪图片进行叠加

addWeighted(drawing,0.5,g\_srcImage,0.5,0.,g\_srcImage);

//定义一个感兴趣的区域

Mat imageROI;

//将裁剪过的图片上关于缺陷检测的数据与原图片叠加

//将感兴趣区域定义在（350，0）处

imageROI = g\_srcImage1(Rect(350, 0, drawing.cols, drawing.rows));

//叠加

addWeighted(imageROI,0.5,drawing,0.5,0.0,imageROI);

//将处理后的图片存储

imwrite("Defect\_map/diaojiao.bmp",g\_srcImage1);

//等待结束

waitKey(0);

}

//压伤程序

void yashang()

{

//局部变量的定义

Mat g\_srcImage;

Mat g\_srcImage1;

Mat g\_grayImage;

Mat threshold\_output;//被减数

Mat threshold\_output\_1;//减数

Mat threshold\_output\_2;//差

vector<vector<Point>> contours;

vector<Vec4i> hierarchy;

//载入原图像

g\_srcImage = imread( "Mosaic\_map/output\_caijian.bmp", 1 );

g\_srcImage1 = imread( "Mosaic\_map/output.bmp", 1 );

//对图像进行灰度处理

cvtColor( g\_srcImage, g\_grayImage, COLOR\_BGR2GRAY );

//使用均值滤波对图像进行降噪blur

blur( g\_grayImage, g\_grayImage, Size(7,7) );

// 减数 使用Threshold检测边缘

threshold( g\_grayImage, threshold\_output\_1,70 , 255, THRESH\_BINARY );

// 被减数 使用Threshold检测边缘

threshold( g\_grayImage, threshold\_output,100 , 255, THRESH\_BINARY );

// 0：矩形 1：十字交叉形 2： 椭圆

Mat element=getStructuringElement(MORPH\_RECT,Size(3,3));

// 0：矩形 1：十字交叉形 2： 椭圆

Mat element1=getStructuringElement(MORPH\_RECT,Size(9,9));

//用（9，9）算子膨胀

erode(threshold\_output\_1,threshold\_output\_1,element1);

//求反

threshold\_output=~threshold\_output;

//求反

threshold\_output\_1=~threshold\_output\_1;

//运算

threshold\_output\_2=threshold\_output-threshold\_output\_1;

//用（3，3）算子对缺陷特征膨胀

dilate(threshold\_output\_2,threshold\_output\_2,element);

//imshow("被减数",threshold\_output);//程序检测用，可作为显示

//imshow("减数",threshold\_output\_1);//程序检测用，可作为显示

//imshow("差",threshold\_output\_2);//程序检测用，可作为显示

// 找出轮廓

findContours( threshold\_output\_2, contours, hierarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE, Point(0, 0) );

//局部变量定义——向量变量

vector<vector<Point> > contours\_poly( contours.size() );

vector<Rect> boundRect( contours.size() );

vector<Point2f>center( contours.size() );

vector<float>radius( contours.size() );

// 多边形逼近轮廓 + 获取矩形和圆形边界框

//一个循环，遍历所有部分

for( unsigned int i = 0; i < contours.size(); i++ )

{

//用指定精度逼近多边形曲线

approxPolyDP( Mat(contours[i]), contours\_poly[i], 3, true );

//计算点集的最外面（up-right）矩形边界

boundRect[i] = boundingRect( Mat(contours\_poly[i]) );

//对给定的 2D点集，寻找最小面积的包围圆形

minEnclosingCircle( contours\_poly[i], center[i], radius[i] );

}

// 绘制

Mat drawing = Mat::zeros( threshold\_output\_2.size(), CV\_8UC3 );

for( int unsigned i = 0; i<contours.size( ); i++ )

{

//设置为粉颜色

Scalar color = Scalar(255,0,255);

//绘制轮廓

drawContours( drawing, contours\_poly, i, color, 2, 8, vector<Vec4i>(), 0, Point() );

//circle( drawing, center[i], (int)radius[i], color, 4, 8, 0 );//绘制圆

addWeighted(g\_srcImage,1,drawing,0.8,0.,g\_srcImage);

//imshow("zui",g\_srcImage);

}

addWeighted(drawing,1,g\_srcImage,0.5,0.,g\_srcImage);

Mat imageROI;

imageROI = g\_srcImage1(Rect(350, 0, drawing.cols, drawing.rows));

addWeighted(imageROI,0.5,drawing,0.5,0.0,imageROI);

imwrite("Defect\_map/huaheng.bmp",g\_srcImage1);

waitKey(0);

}

//图片采集

void CMFC测试Dlg::OnBnClickedButton1()

{

// TODO: 在此添加控件通知处理程序代码

}

//图片裁剪

void CMFC测试Dlg::OnBnClickedButton2()

{

//读取位于Mosaic\_map/output.bmp的图片

Mat image1= imread( "Mosaic\_map/output.bmp");

//定义一个感兴趣区域

Mat imageROI;

//左上角坐标(350,0)到右下角坐标（550，850）的矩形

imageROI=image1(Rect(350,0,550,850));

//imshow("裁剪",imageROI);//显示裁剪区域，调试中可以查看，正式程序中不采用

//等待按下

waitKey(0);

//将处理后的图片写入到Mosaic\_map/output\_caijian.bmp处

imwrite("Mosaic\_map/output\_caijian.bmp",imageROI);

//获取图片路径和图片名称

strFilePath = "Mosaic\_map/output\_caijian.bmp";

CImage image;

image.Load(strFilePath);

//以下两个矩形主要作用是，获取对话框上面的Picture Control的width和height，

//并设置到图片矩形rectPicture，根据图片矩形rectPicture对图片进行处理，

//最后绘制图片到对话框上Picture Control上面

//控件矩形对象

CRect rectControl;

//图片矩形对象

CRect rectPicture;

int x = image.GetWidth();

int y = image.GetHeight();

//Picture Control的ID为IDC\_Src

CWnd \*pWnd = GetDlgItem(IDC\_dst);

pWnd->GetClientRect(rectControl);

CDC \*pDc = GetDlgItem(IDC\_dst)->GetDC();

SetStretchBltMode(pDc->m\_hDC, STRETCH\_HALFTONE);

rectPicture = CRect(rectControl.TopLeft(), CSize((int)rectControl.Width(), (int)rectControl.Height()));

((CStatic\*)GetDlgItem(IDC\_dst))->SetBitmap(NULL);

//以下两种方法都可绘制图片

//将图片绘制到Picture控件表示的矩形区域

//image.StretchBlt(pDc->m\_hDC, rectPicture, SRCCOPY);

//将图片绘制到Picture控件表示的矩形区域

image.Draw(pDc->m\_hDC, rectPicture);

image.Destroy();

pWnd->ReleaseDC(pDc);

}

//轮廓显示

void CMFC测试Dlg::OnBnClickedButton3()

{

// TODO: 在此添加控件通知处理程序代码

lunkuo();

//获取图片路径和图片名称

strFilePath = "Defect\_map/lunkuo.bmp";

CImage image;

image.Load(strFilePath);

//以下两个矩形主要作用是，获取对话框上面的Picture Control的width和height，

//并设置到图片矩形rectPicture，根据图片矩形rectPicture对图片进行处理，

//最后绘制图片到对话框上Picture Control上面

//控件矩形对象

CRect rectControl;

//图片矩形对象

CRect rectPicture;

int x = image.GetWidth();

int y = image.GetHeight();

//Picture Control的ID为IDC\_Src

CWnd \*pWnd = GetDlgItem(IDC\_Src);

pWnd->GetClientRect(rectControl);

CDC \*pDc = GetDlgItem(IDC\_Src)->GetDC();

SetStretchBltMode(pDc->m\_hDC, STRETCH\_HALFTONE);

rectPicture = CRect(rectControl.TopLeft(), CSize((int)rectControl.Width(), (int)rectControl.Height()));

((CStatic\*)GetDlgItem(IDC\_Src))->SetBitmap(NULL);

//以下两种方法都可绘制图片

//将图片绘制到Picture控件表示的矩形区域

//image.StretchBlt(pDc->m\_hDC, rectPicture, SRCCOPY);

//将图片绘制到Picture控件表示的矩形区域

image.Draw(pDc->m\_hDC, rectPicture);

image.Destroy();

pWnd->ReleaseDC(pDc);

}

//缺陷检测

void CMFC测试Dlg::OnBnClickedButton4()

{

// TODO: 在此添加控件通知处理程序代码

Mat dj,hh;

diaojiao();

yashang();

dj=imread("Defect\_map/diaojiao.bmp");

hh=imread("Defect\_map/huaheng.bmp");

addWeighted(dj,0.8,hh,0.8,0.,dj);

imwrite("Defect\_map/zong.bmp",dj);

//获取图片路径和图片名称

strFilePath = "Defect\_map/zong.bmp";

CImage image;

image.Load(strFilePath);

//以下两个矩形主要作用是，获取对话框上面的Picture Control的width和height，

//并设置到图片矩形rectPicture，根据图片矩形rectPicture对图片进行处理，

//最后绘制图片到对话框上Picture Control上面

//控件矩形对象

CRect rectControl;

//图片矩形对象

CRect rectPicture;

int x = image.GetWidth();

int y = image.GetHeight();

//Picture Control的ID为IDC\_Src

CWnd \*pWnd = GetDlgItem(IDC\_Src);

pWnd->GetClientRect(rectControl);

CDC \*pDc = GetDlgItem(IDC\_Src)->GetDC();

SetStretchBltMode(pDc->m\_hDC, STRETCH\_HALFTONE);

rectPicture = CRect(rectControl.TopLeft(), CSize((int)rectControl.Width(), (int)rectControl.Height()));

((CStatic\*)GetDlgItem(IDC\_Src))->SetBitmap(NULL);

//以下两种方法都可绘制图片

//将图片绘制到Picture控件表示的矩形区域

//image.StretchBlt(pDc->m\_hDC, rectPicture, SRCCOPY);

//将图片绘制到Picture控件表示的矩形区域

image.Draw(pDc->m\_hDC, rectPicture);

image.Destroy();

pWnd->ReleaseDC(pDc);

}

//

void CMFC测试Dlg::OnBnClickedButton5()

{

// TODO: 在此添加控件通知处理程序代码

}

//压伤缺陷显示

void CMFC测试Dlg::OnBnClickedButton6()

{

// TODO: 在此添加控件通知处理程序代码

//获取图片路径和图片名称

strFilePath = "Defect\_map/huaheng.bmp";

CImage image;

image.Load(strFilePath);

//以下两个矩形主要作用是，获取对话框上面的Picture Control的width和height，

//并设置到图片矩形rectPicture，根据图片矩形rectPicture对图片进行处理，

//最后绘制图片到对话框上Picture Control上面

//控件矩形对象

CRect rectControl;

//图片矩形对象

CRect rectPicture;

int x = image.GetWidth();

int y = image.GetHeight();

//Picture Control的ID为IDC\_Src

CWnd \*pWnd = GetDlgItem(IDC\_Src);

pWnd->GetClientRect(rectControl);

CDC \*pDc = GetDlgItem(IDC\_Src)->GetDC();

SetStretchBltMode(pDc->m\_hDC, STRETCH\_HALFTONE);

rectPicture = CRect(rectControl.TopLeft(), CSize((int)rectControl.Width(), (int)rectControl.Height()));

((CStatic\*)GetDlgItem(IDC\_Src))->SetBitmap(NULL);

//以下两种方法都可绘制图片

//将图片绘制到Picture控件表示的矩形区域

//image.StretchBlt(pDc->m\_hDC, rectPicture, SRCCOPY);

//将图片绘制到Picture控件表示的矩形区域

image.Draw(pDc->m\_hDC, rectPicture);

image.Destroy();

pWnd->ReleaseDC(pDc);

}

// 掉角缺陷显示

void CMFC测试Dlg::OnBnClickedButton7()

{

//获取图片路径和图片名称

strFilePath = "Defect\_map/diaojiao.bmp";

CImage image;

image.Load(strFilePath);

//以下两个矩形主要作用是，获取对话框上面的Picture Control的width和height，

//并设置到图片矩形rectPicture，根据图片矩形rectPicture对图片进行处理，

//最后绘制图片到对话框上Picture Control上面

//控件矩形对象

CRect rectControl;

//图片矩形对象

CRect rectPicture;

int x = image.GetWidth();

int y = image.GetHeight();

//Picture Control的ID为IDC\_Src

CWnd \*pWnd = GetDlgItem(IDC\_Src);

pWnd->GetClientRect(rectControl);

CDC \*pDc = GetDlgItem(IDC\_Src)->GetDC();

SetStretchBltMode(pDc->m\_hDC, STRETCH\_HALFTONE);

rectPicture = CRect(rectControl.TopLeft(), CSize((int)rectControl.Width(), (int)rectControl.Height()));

((CStatic\*)GetDlgItem(IDC\_Src))->SetBitmap(NULL);

//以下两种方法都可绘制图片

//将图片绘制到Picture控件表示的矩形区域

//image.StretchBlt(pDc->m\_hDC, rectPicture, SRCCOPY);

//将图片绘制到Picture控件表示的矩形区域

image.Draw(pDc->m\_hDC, rectPicture);

image.Destroy();

pWnd->ReleaseDC(pDc);

CString str;

GetDlgItem(IDC\_InputChar)->GetWindowText(str);

ShellExecute(NULL, \_T("open"), \_T("diaojiao\_canshu.txt"), NULL, NULL, SW\_SHOW);

}

//选择图片

void CMFC测试Dlg::OnBnClickedButton8()

{

//参考 https://blog.csdn.net/XuePiaoFei1/article/details/78071471

//选择图片

CFileDialog fileDlg(TRUE, \_T("png"), NULL, 0, \_T("image Files(\*.bmp; \*.jpg;\*.png)|\*.JPG;\*.PNG;\*.BMP|All Files (\*.\*) |\*.\*||"), this);

fileDlg.DoModal();

//获取图片路径和图片名称

//路径

strFilePath = fileDlg.GetPathName();

//名称

strFileName = fileDlg.GetFileName();

//判断路径不为空

if (strFilePath == \_T(""))

{

return;

}

CImage image;

image.Load(strFilePath);//载入

//以下两个矩形主要作用是，获取对话框上面的Picture Control的width和height，

//并设置到图片矩形rectPicture，根据图片矩形rectPicture对图片进行处理，

//最后绘制图片到对话框上Picture Control上面

//控件矩形对象

CRect rectControl;

//图片矩形对象

CRect rectPicture;

//定义x为宽度

int x = image.GetWidth();

//定义y为高度

int y = image.GetHeight();

//Picture Control的ID为IDC\_Src

CWnd \*pWnd = GetDlgItem(IDC\_Src);

pWnd->GetClientRect(rectControl);

CDC \*pDc = GetDlgItem(IDC\_Src)->GetDC();

SetStretchBltMode(pDc->m\_hDC, STRETCH\_HALFTONE);

rectPicture = CRect(rectControl.TopLeft(), CSize((int)rectControl.Width(), (int)rectControl.Height()));

((CStatic\*)GetDlgItem(IDC\_Src))->SetBitmap(NULL);

//以下两种方法都可绘制图片

//将图片绘制到Picture控件表示的矩形区域

//image.StretchBlt(pDc->m\_hDC, rectPicture, SRCCOPY);

//将图片绘制到Picture控件表示的矩形区域

image.Draw(pDc->m\_hDC, rectPicture);

image.Destroy();

pWnd->ReleaseDC(pDc);

//------------------------------------------------------------------------

//参考 https://blog.csdn.net/a478257115/article/details/53031799

//保存

CWnd\* bmpShow = GetDlgItem(IDC\_Src);

CDC \*pdc = bmpShow->GetDC();

CImage imag;

CRect rect;

CString picturePath;

//获取画布大小

GetClientRect(&rect);

bmpShow->GetWindowRect(&rect);

imag.Create(rect.Width(), rect.Height(), 32);

::BitBlt(imag.GetDC(), 0, 0, rect.Width(), rect.Height(), pdc->m\_hDC, 0, 0, SRCCOPY);

//文件路径

picturePath = "Mosaic\_map/output.bmp";

//保存图片

HRESULT hResult = imag.Save(picturePath);

ReleaseDC(pdc);

imag.ReleaseDC();

}

void CMFC测试Dlg::OnEnChangeInputchar()

{

// TODO: 如果该控件是 RICHEDIT 控件，它将不

// 发送此通知，除非重写 CDialogEx::OnInitDialog()

// 函数并调用 CRichEditCtrl().SetEventMask()，

// 同时将 ENM\_CHANGE 标志“或”运算到掩码中。

// TODO: 在此添加控件通知处理程序代码

}

// MFC测试.cpp : 定义应用程序的类行为。

//

#include "stdafx.h"

#include "MFC测试.h"

#include "MFC测试Dlg.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CMFC测试App

BEGIN\_MESSAGE\_MAP(CMFC测试App, CWinApp)

ON\_COMMAND(ID\_HELP, &CWinApp::OnHelp)

END\_MESSAGE\_MAP()

// CMFC测试App 构造

CMFC测试App::CMFC测试App()

{

// 支持重新启动管理器

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

// 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 CMFC测试App 对象

CMFC测试App theApp;

// CMFC测试App 初始化

BOOL CMFC测试App::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);

CWinApp::InitInstance();

AfxEnableControlContainer();

// 创建 shell 管理器，以防对话框包含

// 任何 shell 树视图控件或 shell 列表视图控件。

CShellManager \*pShellManager = new CShellManager;

// 激活“Windows Native”视觉管理器，以便在 MFC 控件中启用主题

CMFCVisualManager::SetDefaultManager(RUNTIME\_CLASS(CMFCVisualManagerWindows));

// 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"));

CMFC测试Dlg dlg;

m\_pMainWnd = &dlg;

INT\_PTR nResponse = dlg.DoModal();

if (nResponse == IDOK)

{

// TODO: 在此放置处理何时用

// “确定”来关闭对话框的代码

}

else if (nResponse == IDCANCEL)

{

// TODO: 在此放置处理何时用

// “取消”来关闭对话框的代码

}

else if (nResponse == -1)

{

TRACE(traceAppMsg, 0, "警告: 对话框创建失败，应用程序将意外终止。\n");

TRACE(traceAppMsg, 0, "警告: 如果您在对话框上使用 MFC 控件，则无法 #define \_AFX\_NO\_MFC\_CONTROLS\_IN\_DIALOGS。\n");

}

// 删除上面创建的 shell 管理器。

if (pShellManager != NULL)

{

delete pShellManager;

}

// 由于对话框已关闭，所以将返回 FALSE 以便退出应用程序，

// 而不是启动应用程序的消息泵。

return FALSE;

}

// stdafx.cpp : 只包括标准包含文件的源文件

// MFC测试.pch 将作为预编译头

// stdafx.obj 将包含预编译类型信息

#include "stdafx.h"

#pragma once

// 包括 SDKDDKVer.h 将定义最高版本的可用 Windows 平台。

// 如果要为以前的 Windows 平台生成应用程序，请包括 WinSDKVer.h，并将

// WIN32\_WINNT 宏设置为要支持的平台，然后再包括 SDKDDKVer.h。

#include <SDKDDKVer.h>

// stdafx.h : 标准系统包含文件的包含文件，

// 或是经常使用但不常更改的

// 特定于项目的包含文件

#pragma once

#ifndef VC\_EXTRALEAN

#define VC\_EXTRALEAN // 从 Windows 头中排除极少使用的资料

#endif

#include "targetver.h"

#define \_ATL\_CSTRING\_EXPLICIT\_CONSTRUCTORS // 某些 CString 构造函数将是显式的

// 关闭 MFC 对某些常见但经常可放心忽略的警告消息的隐藏

#define \_AFX\_ALL\_WARNINGS

#include <afxwin.h> // MFC 核心组件和标准组件

#include <afxext.h> // MFC 扩展

#include <afxdisp.h> // MFC 自动化类

#ifndef \_AFX\_NO\_OLE\_SUPPORT

#include <afxdtctl.h> // MFC 对 Internet Explorer 4 公共控件的支持

#endif

#ifndef \_AFX\_NO\_AFXCMN\_SUPPORT

#include <afxcmn.h> // MFC 对 Windows 公共控件的支持

#endif // \_AFX\_NO\_AFXCMN\_SUPPORT

#include <afxcontrolbars.h> // 功能区和控件条的 MFC 支持

#ifdef \_UNICODE

#if defined \_M\_IX86

#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='x86' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#elif defined \_M\_X64

#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='amd64' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#else

#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#endif

#endif

//{{NO\_DEPENDENCIES}}

// Microsoft Visual C++ 生成的包含文件。

// 供 MFC测试.rc 使用

//

#define IDM\_ABOUTBOX 0x0010

#define IDD\_ABOUTBOX 100

#define IDS\_ABOUTBOX 101

#define IDD\_MFC\_DIALOG 102

#define IDR\_MAINFRAME 128

#define IDC\_BUTTON1 1000

#define IDC\_BUTTON2 1001

#define IDC\_BUTTON3 1002

#define IDC\_BUTTON4 1003

#define IDC\_BUTTON5 1004

#define IDC\_BUTTON6 1005

#define IDC\_EDIT2 1006

#define IDC\_EDIT3 1007

#define IDC\_InputChar 1007

#define IDC\_BUTTON7 1008

#define IDC\_Src 1009

#define IDC\_BUTTON10 1010

#define IDC\_BUTTON8 1011

#define IDC\_BUTTON9 1012

#define IDC\_dst 1013

// Next default values for new objects

//

#ifdef APSTUDIO\_INVOKED

#ifndef APSTUDIO\_READONLY\_SYMBOLS

#define \_APS\_NEXT\_RESOURCE\_VALUE 129

#define \_APS\_NEXT\_COMMAND\_VALUE 32771

#define \_APS\_NEXT\_CONTROL\_VALUE 1015

#define \_APS\_NEXT\_SYMED\_VALUE 101

#endif

#endif

// MFC测试Dlg.h : 头文件

#pragma once

// CMFC测试Dlg 对话框

class CMFC测试Dlg : public CDialogEx

{

// 构造

public:

CMFC测试Dlg(CWnd\* pParent = NULL); // 标准构造函数

// 对话框数据

enum { IDD = IDD\_MFC\_DIALOG };

protected:

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

// 实现

protected:

HICON m\_hIcon;

// 生成的消息映射函数

virtual BOOL OnInitDialog();

afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);

afx\_msg void OnPaint();

afx\_msg HCURSOR OnQueryDragIcon();

DECLARE\_MESSAGE\_MAP()

public:

afx\_msg void OnBnClickedButton1();

afx\_msg void OnBnClickedButton2();

afx\_msg void OnBnClickedButton3();

afx\_msg void OnEnChangeEdit1();

afx\_msg void OnBnClickedButton5();

afx\_msg void OnBnClickedButton8();

afx\_msg void OnBnClickedButton4();

afx\_msg void OnBnClickedButton7();

afx\_msg void OnBnClickedButton6();

afx\_msg void OnBnClickedButton9();

afx\_msg void OnEnChangeInputchar();

};

// MFC测试.h : PROJECT\_NAME 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

// CMFC测试App:

// 有关此类的实现，请参阅 MFC测试.cpp

//

class CMFC测试App : public CWinApp

{

public:

CMFC测试App();

// 重写

public:

virtual BOOL InitInstance();

// 实现

DECLARE\_MESSAGE\_MAP()

};

extern CMFC测试App theApp;