## H文件

// Image.h: interface for the CImage class.

//

//////////////////////////////////////////////////////////////////////

#if !defined(AFX\_IMAGE\_H\_\_377F5446\_CBF5\_4179\_8D56\_463F20CD3A86\_\_INCLUDED\_)

#define AFX\_IMAGE\_H\_\_377F5446\_CBF5\_4179\_8D56\_463F20CD3A86\_\_INCLUDED\_

#if \_MSC\_VER > 1000

#pragma once

#endif // \_MSC\_VER > 1000

#include "Include.h"

#include "TMatrix.h"

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ]

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* << 图像类 >> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ]

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ]

class CImage

{

// [ \*\*\*\*\*\*\*\* ] ......................................................

// [ 内部类型 ] ......................................................

// [ \*\*\*\*\*\*\*\* ] ......................................................

public:

enum ImageType{ Color, Gray, Unicolor, NoImage }; // [ 图像类型 ]

// [ \*\*\*\*\*\*\*\* ] ......................................................

// [ 成员变量 ] ......................................................

// [ \*\*\*\*\*\*\*\* ] ......................................................

private:

ImageType m\_imageType; // [ 图像类型 ]

CTMatrix< RGB\_TRIPLE > m\_colorImage; // [ 彩色图像数据 ]

CTMatrix< BYTE > m\_grayImage; // [ 灰度图像数据 ]

CTMatrix< BlackWhite> m\_unicolorImage; // [ 二值图像数据 ]

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ............................................

// [ 构造函数和析构函数 ] ............................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ............................................

public:

CImage(); // [ 缺省构造函数 ]

CImage( const CImage& image ); // [ 复制构造函数 ]

virtual ~CImage(); // [ 析构函数 ]

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 功能接口函数 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

public:

void ImportFrom( const CString& fileName ); // [ 导入图像文件 ]

void ImportFrom( const CImage& image ); // [ 导入图像数据 ]

void ImportFrom( const CTMatrix< RGB\_TRIPLE >& color\_image ); // [ 导入彩色图像数据 ]

void ImportFrom( const CTMatrix< BYTE >& gray\_image ); // [ 导入灰度图像数据 ]

void ImportFrom( const CTMatrix< BlackWhite >& unicolor\_image ); // [ 导入二值图像数据 ]

void ImportFrom( CBitmap& bitmap ); // [ 导入设备相关位图 ]

void ImportFrom( HANDLE hDIB ); // [ 导入设备无关位图 ]

CImage& operator= ( const CImage& image ); // [ 重载赋值操作符 ]

void ShowImage( CDC\* pDC, CPoint leftTop, CSize displaySize = CSize( 0, 0 ) ) const; // [ 显示图像 ]

void SaveImage( const CString& fileName, int jpeg\_quality = 100 ) const; // [ 保存图像 ]

static void ShowImage( const CTMatrix< RGB\_TRIPLE >& colorImage,

CDC\* pDC, CPoint leftTop, CSize displaySize = CSize( 0, 0 ) ); // [ 显示彩色图像 ]

static void ShowImage( const CTMatrix< BYTE >& grayImage,

CDC\* pDC, CPoint leftTop, CSize displaySize = CSize( 0, 0 ) ); // [ 显示灰度图像 ]

static void ShowImage( const CTMatrix< BlackWhite >& unicolorImage,

CDC\* pDC, CPoint leftTop, CSize displaySize = CSize( 0, 0 ) ); // [ 显示二值图像 ]

static void SaveImage( const CTMatrix< RGB\_TRIPLE >& colorImage, const CString& fileName, int jpeg\_quality ) ; // [ 保存彩色图像 ]

static void SaveImage( const CTMatrix< BYTE >& grayImage, const CString& fileName, int jpeg\_quality ) ; // [ 保存灰度图像 ]

static void SaveImage( const CTMatrix< BlackWhite >& unicolorImage, const CString& fileName ); // [ 保存二值图像 ]

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 状态参数函数 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

public:

ImageType GetImageType() const; // [ 图像类型 ]

bool IsColorImage() const; // [ 是否是彩色图像 ]

bool IsGrayImage() const; // [ 是否是灰度图像 ]

bool IsUnicolorImage() const; // [ 是否是二值图像 ]

bool IsNull() const; // [ 是否为空 ]

const CTMatrix< RGB\_TRIPLE >& Get\_color\_image() const; // [ 彩色图像数据 ]

const CTMatrix< BYTE >& Get\_gray\_image() const; // [ 灰度图像数据 ]

const CTMatrix< BlackWhite >& Get\_unicolor\_image() const; // [ 二值图像数据 ]

long Get\_image\_height() const; // [ 图像高度 ]

long Get\_image\_width() const; // [ 图像宽度 ]

CImagePoint Get\_image\_size() const; // [ 图像尺寸 ]

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 内部实现函数 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

private:

static CString PostfixToFileName( const CString& fileName ); // [ 获取文件名后缀 ]

void ImportFromBmpFile( const CString& fileName ); // [ 导入 BMP 图像文件 ]

void ImportFrom\_32\_BmpFile( BYTE\* bitmapData, bool isBottomUp,

long imageHeight, long imageWidth,

long bytesPerMemoryRow, long bytesPerFileRow ); // [ 导入 32 位 BMP 图像数据 ]

void ImportFrom\_24\_BmpFile( BYTE\* bitmapData, bool isBottomUp,

long imageHeight, long imageWidth,

long bytesPerMemoryRow, long bytesPerFileRow ); // [ 导入 24 位 BMP 图像数据 ]

void ImportFrom\_8\_BmpFile( BYTE\* bitmapData, RGBQUAD\* colorTemplate, bool isBottomUp,

long imageHeight, long imageWidth,

long bytesPerMemoryRow, long bytesPerFileRow ); // [ 导入 8 位 BMP 图像数据 ]

void ImportFrom\_1\_BmpFile( BYTE\* bitmapData, RGBQUAD\* colorTemplate, bool isBottomUp,

long imageHeight, long imageWidth,

long bytesPerMemoryRow, long bytesPerFileRow ); // [ 导入 1 位 BMP 图像数据 ]

void Import\_from\_raw\_file( const CString& file\_name ); // [ 导入 RAW 图像文件 ]

HANDLE DDBToDIB( CBitmap& bitmap, DWORD dwCompression, CPalette\* pPal ); // [ 设备相关位图转换为设备无关位图 ]

static void SaveImageToBmpFile( const CTMatrix< RGB\_TRIPLE >& colorImage, const CString& fileName ); // [ 保存彩色图像为 BMP 文件 ]

static void SaveImageToBmpFile( const CTMatrix< BYTE >& grayImage, const CString& fileName ); // [ 保存灰度图像为 BMP 文件 ]

static void SaveImageToBmpFile( const CTMatrix< BlackWhite >& unicolorImage, const CString& fileName ); // [ 保存二值图像为 BMP 文件 ]

static void SaveImageToJpegFile( const CTMatrix< RGB\_TRIPLE >& colorImage, const CString& fileName, int jpeg\_quality ); // [ 保存彩色图像为 JPEG 文件 ]

static void SaveImageToJpegFile( const CTMatrix< BYTE >& grayImage, const CString& fileName, int jpeg\_quality ); // [ 保存灰度图像为 JPEG 文件 ]

static void Construct\_1\_Info( BITMAPINFO\* bitmapInfo, long imageHeight, long imageWidth ); // [ 构造 1 位信息 ]

static void Construct\_1\_FileHeader( BITMAPFILEHEADER\* fileHeader, long imageHeight, long imageWidth ); // [ 构造 1 位文件头 ]

static void Construct\_8\_Info( BITMAPINFO\* bitmapInfo, long imageHeight, long imageWidth ); // [ 构造 8 位信息 ]

static void Construct\_8\_FileHeader( BITMAPFILEHEADER\* fileHeader, long imageHeight, long imageWidth ); // [ 构造 8 位文件头 ]

static void Construct\_24\_Info( BITMAPINFO\* bitmapInfo, long imageHeight, long imageWidth ); // [ 构造 24 位信息 ]

static void Construct\_24\_FileHeader( BITMAPFILEHEADER\* fileHeader, long imageHeight, long imageWidth ); // [ 构造 24 位文件头 ]

static void Construct\_32\_Info( BITMAPINFO\* bitmapInfo, long imageHeight, long imageWidth ); // [ 构造 32 位信息 ]

};

#endif // !defined(AFX\_IMAGE\_H\_\_377F5446\_CBF5\_4179\_8D56\_463F20CD3A86\_\_INCLUDED\_)

## CPP文件

// Image.cpp: implementation of the CImage class.

//

//////////////////////////////////////////////////////////////////////

#include "stdafx.h"

#include "Image.h"

#ifdef \_DEBUG

#undef THIS\_FILE

static char THIS\_FILE[]=\_\_FILE\_\_;

#define new DEBUG\_NEW

#endif

//////////////////////////////////////////////////////////////////////

// Construction/Destruction

//////////////////////////////////////////////////////////////////////

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 缺省构造函数 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

CImage::CImage()

{

m\_imageType = NoImage; // [ 初始化图像类型 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 复制构造函数 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

CImage::CImage( const CImage& image )

{

m\_imageType = NoImage; // [ 初始化图像类型 ]

ImportFrom( image ); // [ 导入图像数据 ]

}

// [ \*\*\*\*\*\*\*\* ] ..........................................................

// [ 析构函数 ] ..........................................................

// [ \*\*\*\*\*\*\*\* ] ..........................................................

CImage::~CImage()

{

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

// [ 重载赋值操作符 ] ....................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

CImage& CImage::operator= ( const CImage& image )

{

ImportFrom( image ); // [ 导入图像数据 ]

return \* this; // [ 返回当前对象 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 导入图像文件 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

void CImage::ImportFrom( const CString& fileName )

{

m\_imageType = NoImage; // [ 初始化图像类型 ]

CString postfix = PostfixToFileName( fileName ); // [ 获取文件名后缀 ]

if( postfix.CompareNoCase( "BMP" ) == 0 ) // [ BMP 图像 ]

{

ImportFromBmpFile( fileName ); // [ 导入 BMP 图像文件 ]

return; // [ 结束导入 ]

}

if( postfix.CompareNoCase( "RAW" ) == 0 ) // [ 如果是 RAW 图像格式，那么 ... ]

{

Import\_from\_raw\_file( fileName ); // [ 导入 RAW 图像文件 ]

return; // [ 结束导入 ]

}

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 导入图像数据 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

void CImage::ImportFrom( const CImage& image )

{

m\_imageType = image.m\_imageType; // [ 图像类型 ]

m\_colorImage = image.m\_colorImage; // [ 彩色图像数据 ]

m\_grayImage = image.m\_grayImage; // [ 灰度图像数据 ]

m\_unicolorImage = image.m\_unicolorImage; // [ 二值图像数据 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 导入彩色图像数据 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

void CImage::ImportFrom( const CTMatrix< RGB\_TRIPLE >& color\_image )

{

m\_imageType = Color; // [ 图像类型 ]

m\_colorImage = color\_image; // [ 图像数据 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 导入灰度图像数据 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

void CImage::ImportFrom( const CTMatrix< BYTE >& gray\_image )

{

m\_imageType = Gray; // [ 图像类型 ]

m\_grayImage = gray\_image; // [ 图像数据 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 导入二值图像数据 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

void CImage::ImportFrom( const CTMatrix< BlackWhite >& unicolor\_image )

{

m\_imageType = Unicolor; // [ 图像类型 ]

m\_unicolorImage = unicolor\_image; // [ 图像数据 ]

}

// [ \*\*\*\*\*\*\*\* ] ..........................................................

// [ 显示图像 ] ..........................................................

// [ \*\*\*\*\*\*\*\* ] ..........................................................

void CImage::ShowImage( CDC\* pDC, CPoint leftTop, CSize displaySize ) const

{

switch( m\_imageType ) // [ 图像类型 ]

{

case Color: // [ 彩色图像 ]

ShowImage( m\_colorImage, pDC, leftTop, displaySize ); // [ 显示彩色图像 ]

break;

case Gray: // [ 灰度图像 ]

ShowImage( m\_grayImage, pDC, leftTop, displaySize ); // [ 显示灰度图像 ]

break;

case Unicolor: // [ 二值图像 ]

ShowImage( m\_unicolorImage, pDC, leftTop, displaySize ); // [ 显示二值图像 ]

break;

}

}

// [ \*\*\*\*\*\*\*\* ] ..........................................................

// [ 保存图像 ] ..........................................................

// [ \*\*\*\*\*\*\*\* ] ..........................................................

void CImage::SaveImage( const CString& fileName, int jpeg\_quality ) const

{

switch( m\_imageType ) // [ 图像类型 ]

{

case Color: // [ 彩色图像 ]

SaveImage( m\_colorImage, fileName, jpeg\_quality ); // [ 保存彩色图像 ]

break;

case Gray: // [ 灰度图像 ]

SaveImage( m\_grayImage, fileName, jpeg\_quality ); // [ 保存灰度图像 ]

break;

case Unicolor: // [ 二值图像 ]

SaveImage( m\_unicolorImage, fileName ); // [ 保存二值图像 ]

break;

}

}

// [ \*\*\*\*\*\*\*\* ] ..........................................................

// [ 图像类型 ] ..........................................................

// [ \*\*\*\*\*\*\*\* ] ..........................................................

CImage::ImageType CImage::GetImageType() const

{

return m\_imageType;

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 彩色图像数据 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

const CTMatrix< RGB\_TRIPLE >& CImage::Get\_color\_image() const

{

#ifdef \_DEBUG

if( m\_imageType != Color )

{

AfxMessageBox( "Program Error << Image Type Error >>" );

}

#endif // \_DEBUG

return m\_colorImage;

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 灰度图像数据 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

const CTMatrix< BYTE >& CImage::Get\_gray\_image() const

{

#ifdef \_DEBUG

if( m\_imageType != Gray )

{

AfxMessageBox( "Program Error << Image Type Error >>" );

}

#endif // \_DEBUG

return m\_grayImage;

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 二值图像数据 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

const CTMatrix< BlackWhite >& CImage::Get\_unicolor\_image() const

{

#ifdef \_DEBUG

if( m\_imageType != Unicolor )

{

AfxMessageBox( "Program Error << Image Type Error >>" );

}

#endif // \_DEBUG

return m\_unicolorImage;

}

// [ \*\*\*\*\*\*\*\* ] ..........................................................

// [ 图像高度 ] ..........................................................

// [ \*\*\*\*\*\*\*\* ] ..........................................................

long CImage::Get\_image\_height() const

{

switch( m\_imageType ) // [ 图像类型 ]

{

case Color: // [ 彩色图像 ]

return m\_colorImage.GetRows();

break;

case Gray: // [ 灰度图像 ]

return m\_grayImage.GetRows();

break;

case Unicolor: // [ 二值图像 ]

return m\_unicolorImage.GetRows();

break;

default: // [ 空图像 ]

return 0;

break;

}

}

// [ \*\*\*\*\*\*\*\* ] ..........................................................

// [ 图像宽度 ] ..........................................................

// [ \*\*\*\*\*\*\*\* ] ..........................................................

long CImage::Get\_image\_width() const

{

switch( m\_imageType ) // [ 图像类型 ]

{

case Color: // [ 彩色图像 ]

return m\_colorImage.GetColumns();

break;

case Gray: // [ 灰度图像 ]

return m\_grayImage.GetColumns();

break;

case Unicolor: // [ 二值图像 ]

return m\_unicolorImage.GetColumns();

break;

default: // [ 空图像 ]

return 0;

break;

}

}

// [ \*\*\*\*\*\*\*\* ] ..........................................................

// [ 图像尺寸 ] ..........................................................

// [ \*\*\*\*\*\*\*\* ] ..........................................................

CImagePoint CImage::Get\_image\_size() const

{

switch( m\_imageType ) // [ 图像类型 ]

{

case Color: // [ 彩色图像 ]

return m\_colorImage.Get\_size();

break;

case Gray: // [ 灰度图像 ]

return m\_grayImage.Get\_size();

break;

case Unicolor: // [ 二值图像 ]

return m\_unicolorImage.Get\_size();

break;

default: // [ 空图像 ]

return CImagePoint( 0, 0 );

break;

}

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

// [ 是否是彩色图像 ] ....................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

bool CImage::IsColorImage() const

{

return ( m\_imageType == Color );

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

// [ 是否是灰度图像 ] ....................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

bool CImage::IsGrayImage() const

{

return ( m\_imageType == Gray );

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

// [ 是否是二值图像 ] ....................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

bool CImage::IsUnicolorImage() const

{

return ( m\_imageType == Unicolor );

}

// [ \*\*\*\*\*\*\*\* ] ..........................................................

// [ 是否为空 ] ..........................................................

// [ \*\*\*\*\*\*\*\* ] ..........................................................

bool CImage::IsNull() const

{

return ( m\_imageType == NoImage );

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

// [ 获取文件名后缀 ] ....................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

CString CImage::PostfixToFileName( const CString& fileName )

{

CString postfix; // [ 文件名后缀 ]

int position = fileName.ReverseFind('.'); // [ 文件名中最个一个点字符的位置 ]

if( position != -1 ) // [ 文件名是否存在后缀 ]

{

postfix = fileName.Right( fileName.GetLength() - position - 1 ); // [ 获取文件名后缀 ]

postfix.MakeUpper(); // [ 大写文件名后缀 ]

}

return postfix; // [ 返回文件名后缀 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................

// [ 设备相关位图转换为设备无关位图 ] ....................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................

//

// DDBToDIB - Creates a DIB from a DDB

// bitmap - Device dependent bitmap

// dwCompression - Type of compression - see BITMAPINFOHEADER

// pPal - Logical palette

//

HANDLE CImage::DDBToDIB( CBitmap& bitmap, DWORD dwCompression, CPalette\* pPal )

{

BITMAP bm;

BITMAPINFOHEADER bi;

LPBITMAPINFOHEADER lpbi;

DWORD dwLen;

HANDLE hDIB;

HANDLE handle;

HDC hDC;

HPALETTE hPal;

ASSERT( bitmap.GetSafeHandle() );

// The function has no arg for bitfields

if( dwCompression == BI\_BITFIELDS )

return NULL;

// If a palette has not been supplied use defaul palette

hPal = (HPALETTE) pPal->GetSafeHandle();

if (hPal==NULL)

hPal = (HPALETTE) GetStockObject(DEFAULT\_PALETTE);

// Get bitmap information

bitmap.GetObject(sizeof(bm),(LPSTR)&bm);

// Initialize the bitmapinfoheader

bi.biSize = sizeof(BITMAPINFOHEADER);

bi.biWidth = bm.bmWidth;

bi.biHeight = bm.bmHeight;

bi.biPlanes = 1;

bi.biBitCount = bm.bmPlanes \* bm.bmBitsPixel;

bi.biCompression = dwCompression;

bi.biSizeImage = 0;

bi.biXPelsPerMeter = 0;

bi.biYPelsPerMeter = 0;

bi.biClrUsed = 0;

bi.biClrImportant = 0;

// Compute the size of the infoheader and the color table

int nColors = (1 << bi.biBitCount);

if( nColors > 256 )

nColors = 0;

dwLen = bi.biSize + nColors \* sizeof(RGBQUAD);

// We need a device context to get the DIB from

hDC = GetDC(NULL);

hPal = SelectPalette(hDC,hPal,FALSE);

RealizePalette(hDC);

// Allocate enough memory to hold bitmapinfoheader and color table

hDIB = GlobalAlloc(GMEM\_FIXED,dwLen);

if (!hDIB){

SelectPalette(hDC,hPal,FALSE);

ReleaseDC(NULL,hDC);

return NULL;

}

lpbi = (LPBITMAPINFOHEADER)hDIB;

\*lpbi = bi;

// Call GetDIBits with a NULL lpBits param, so the device driver

// will calculate the biSizeImage field

GetDIBits(hDC, (HBITMAP)bitmap.GetSafeHandle(), 0L, (DWORD)bi.biHeight,

(LPBYTE)NULL, (LPBITMAPINFO)lpbi, (DWORD)DIB\_RGB\_COLORS);

bi = \*lpbi;

// If the driver did not fill in the biSizeImage field, then compute it

// Each scan line of the image is aligned on a DWORD (32bit) boundary

if (bi.biSizeImage == 0){

bi.biSizeImage = ((((bi.biWidth \* bi.biBitCount) + 31) & ~31) / 8)

\* bi.biHeight;

// If a compression scheme is used the result may infact be larger

// Increase the size to account for this.

if (dwCompression != BI\_RGB)

bi.biSizeImage = (bi.biSizeImage \* 3) / 2;

}

// Realloc the buffer so that it can hold all the bits

dwLen += bi.biSizeImage;

if (handle = GlobalReAlloc(hDIB, dwLen, GMEM\_MOVEABLE))

hDIB = handle;

else{

GlobalFree(hDIB);

// Reselect the original palette

SelectPalette(hDC,hPal,FALSE);

ReleaseDC(NULL,hDC);

return NULL;

}

// Get the bitmap bits

lpbi = (LPBITMAPINFOHEADER)hDIB;

// FINALLY get the DIB

BOOL bGotBits = GetDIBits( hDC, (HBITMAP)bitmap.GetSafeHandle(),

0L, // Start scan line

(DWORD)bi.biHeight, // # of scan lines

(LPBYTE)lpbi // address for bitmap bits

+ (bi.biSize + nColors \* sizeof(RGBQUAD)),

(LPBITMAPINFO)lpbi, // address of bitmapinfo

(DWORD)DIB\_RGB\_COLORS); // Use RGB for color table

if( !bGotBits )

{

GlobalFree(hDIB);

SelectPalette(hDC,hPal,FALSE);

ReleaseDC(NULL,hDC);

return NULL;

}

SelectPalette(hDC,hPal,FALSE);

ReleaseDC(NULL,hDC);

return hDIB;

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 导入设备相关位图 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

void CImage::ImportFrom( CBitmap& bitmap )

{

m\_imageType = NoImage; // [ 初始化图像类型 ]

// Convert the bitmap to a DIB

HANDLE hDIB = DDBToDIB( bitmap, BI\_RGB, NULL );

if( hDIB == NULL ) return;

// [ 导入设备无关位图 ]

ImportFrom( hDIB );

// Free the memory allocated by DDBToDIB for the DIB

GlobalFree( hDIB );

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 导入设备无关位图 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

void CImage::ImportFrom( HANDLE hDIB )

{

BITMAPFILEHEADER hdr;

LPBITMAPINFOHEADER lpbi;

if (!hDIB) return;

lpbi = (LPBITMAPINFOHEADER)hDIB;

int nColors = 1 << lpbi->biBitCount;

if( nColors > 256 )

nColors = 0;

// Fill in the fields of the file header

hdr.bfType = ((WORD) ('M' << 8) | 'B'); // is always "BM"

hdr.bfSize = DWORD( GlobalSize (hDIB) + sizeof( hdr ) );

hdr.bfReserved1 = 0;

hdr.bfReserved2 = 0;

hdr.bfOffBits = (DWORD) (sizeof( hdr ) + lpbi->biSize +

nColors \* sizeof(RGBQUAD));

//////////////////////////////////////////////////////////////////////

BITMAPFILEHEADER\* bitmapFileHeader; // [ 文件头指针 ]

BITMAPINFOHEADER\* bitmapInfoHeader; // [ 信息头指针 ]

BITMAPINFO\* bitmapInfo; // [ 信息指针 ]

RGBQUAD\* colorTemplate; // [ 调色板指针 ]

BYTE\* bitmapData; // [ 数据指针 ]

bitmapFileHeader = &hdr; // [ 设置文件头指针 ]

bitmapInfoHeader = lpbi; // [ 设置信息头指针 ]

bitmapInfo = ( BITMAPINFO\* )bitmapInfoHeader; // [ 设置信息指针 ]

colorTemplate = ( RGBQUAD\* )bitmapInfo->bmiColors; // [ 设置调色板指针 ]

bitmapData = (BYTE\*)hDIB + bitmapFileHeader->bfOffBits - sizeof( hdr ); // [ 设置数据指针 ]

if( bitmapFileHeader->bfType != ((WORD) ('M' << 8) | 'B') ) // [ 检查 BMP 文件的合法性 ]

{

return; // [ 函数返回 ]

}

long imageWidth = bitmapInfoHeader->biWidth; // [ 图像宽度( 像素 ) ]

long imageHeight = bitmapInfoHeader->biHeight; // [ 图像高度( 像素 ) ]

imageHeight = ( imageHeight > 0 ) ? imageHeight : - imageHeight ; // [ 修正图像高度 ]

bool isBottomUp = bitmapInfoHeader->biHeight > 0; // [ 图像存储顺序 ]

long bitCount = bitmapInfoHeader->biBitCount; // [ 像素存储位数 ]

long bytesPerMemoryRow = ( imageWidth \* bitCount + 7 ) / 8 ; // [ 内存中图像行存储字节数 ]

long bytesPerFileRow = ( ( bytesPerMemoryRow + 3 ) / 4 ) \* 4 ; // [ 文件中图像行存储字节数 ]

switch( bitCount ) // [ 像素存储位数 ]

{

case 1: // [ 二值图像 ]

ImportFrom\_1\_BmpFile( bitmapData, colorTemplate, isBottomUp,

imageHeight, imageWidth,

bytesPerMemoryRow, bytesPerFileRow ); // [ 导入 1 位 BMP 图像数据 ]

break;

case 8: // [ 灰度图像 ]

ImportFrom\_8\_BmpFile( bitmapData, colorTemplate, isBottomUp,

imageHeight, imageWidth,

bytesPerMemoryRow, bytesPerFileRow ); // [ 导入 8 位 BMP 图像数据 ]

break;

case 24: // [ 彩色图像 ]

ImportFrom\_24\_BmpFile( bitmapData, isBottomUp,

imageHeight, imageWidth,

bytesPerMemoryRow, bytesPerFileRow ); // [ 导入 24 位 BMP 图像数据 ]

break;

case 32: // [ 彩色图像 ]

ImportFrom\_32\_BmpFile( bitmapData, isBottomUp,

imageHeight, imageWidth,

bytesPerMemoryRow, bytesPerFileRow ); // [ 导入 32 位 BMP 图像数据 ]

break;

}

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] .................................................

// [ 导入 BMP 图像文件 ] .................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] .................................................

void CImage::ImportFromBmpFile( const CString& fileName )

{

CFile bitmapFile; // [ 图像文件 ]

if( !bitmapFile.Open( fileName, CFile::modeRead ) ) // [ 打开图像文件 ]

return;

long fileLength = long( bitmapFile.GetLength() ); // [ 图像文件长度( 字节数 ) ]

BYTE\* fileBuffer = new BYTE[ fileLength ]; // [ 申请图像文件缓存 ]

bitmapFile.Read( fileBuffer, fileLength ); // [ 读取整个图像文件 ]

BITMAPFILEHEADER\* bitmapFileHeader; // [ 文件头指针 ]

BITMAPINFOHEADER\* bitmapInfoHeader; // [ 信息头指针 ]

BITMAPINFO\* bitmapInfo; // [ 信息指针 ]

RGBQUAD\* colorTemplate; // [ 调色板指针 ]

BYTE\* bitmapData; // [ 数据指针 ]

bitmapFileHeader = ( BITMAPFILEHEADER\* )fileBuffer; // [ 设置文件头指针 ]

bitmapInfoHeader = ( BITMAPINFOHEADER\* )( fileBuffer + sizeof( BITMAPFILEHEADER ) ); // [ 设置信息头指针 ]

bitmapInfo = ( BITMAPINFO\* )bitmapInfoHeader; // [ 设置信息指针 ]

colorTemplate = ( RGBQUAD\* )bitmapInfo->bmiColors; // [ 设置调色板指针 ]

bitmapData = fileBuffer + bitmapFileHeader->bfOffBits; // [ 设置数据指针 ]

if( bitmapFileHeader->bfType != ((WORD) ('M' << 8) | 'B') ) // [ 检查 BMP 文件的合法性 ]

{

delete [] fileBuffer; // [ 释放图像文件缓存 ]

bitmapFile.Close(); // [ 关闭图像文件 ]

return; // [ 函数返回 ]

}

long imageWidth = bitmapInfoHeader->biWidth; // [ 图像宽度( 像素 ) ]

long imageHeight = bitmapInfoHeader->biHeight; // [ 图像高度( 像素 ) ]

imageHeight = ( imageHeight > 0 ) ? imageHeight : - imageHeight ; // [ 修正图像高度 ]

bool isBottomUp = bitmapInfoHeader->biHeight > 0; // [ 图像存储顺序 ]

long bitCount = bitmapInfoHeader->biBitCount; // [ 像素存储位数 ]

long bytesPerMemoryRow = ( imageWidth \* bitCount + 7 ) / 8 ; // [ 内存中图像行存储字节数 ]

long bytesPerFileRow = ( ( bytesPerMemoryRow + 3 ) / 4 ) \* 4 ; // [ 文件中图像行存储字节数 ]

switch( bitCount ) // [ 像素存储位数 ]

{

case 1: // [ 二值图像 ]

ImportFrom\_1\_BmpFile( bitmapData, colorTemplate, isBottomUp,

imageHeight, imageWidth,

bytesPerMemoryRow, bytesPerFileRow ); // [ 导入 1 位 BMP 图像数据 ]

break;

case 8: // [ 灰度图像 ]

ImportFrom\_8\_BmpFile( bitmapData, colorTemplate, isBottomUp,

imageHeight, imageWidth,

bytesPerMemoryRow, bytesPerFileRow ); // [ 导入 8 位 BMP 图像数据 ]

break;

case 24: // [ 彩色图像 ]

ImportFrom\_24\_BmpFile( bitmapData, isBottomUp,

imageHeight, imageWidth,

bytesPerMemoryRow, bytesPerFileRow ); // [ 导入 24 位 BMP 图像数据 ]

break;

case 32: // [ 彩色图像 ]

ImportFrom\_32\_BmpFile( bitmapData, isBottomUp,

imageHeight, imageWidth,

bytesPerMemoryRow, bytesPerFileRow ); // [ 导入 32 位 BMP 图像数据 ]

break;

}

delete [] fileBuffer; // [ 释放图像文件缓存 ]

bitmapFile.Close(); // [ 关闭图像文件 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...........................................

// [ 导入 32 位 BMP 图像数据 ] ...........................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...........................................

void CImage::ImportFrom\_32\_BmpFile( BYTE\* bitmapData, bool isBottomUp,

long imageHeight, long imageWidth,

long bytesPerMemoryRow, long bytesPerFileRow )

{

m\_imageType = Color; // [ 设为彩色图像 ]

m\_colorImage.Construction( imageHeight, imageWidth ); // [ 构造图像 ]

BYTE\* swapBuffer = new BYTE[ imageWidth \* 3 \* imageHeight ]; // [ 申请缓冲区 ]

for( int row = 0; row < imageHeight; row++ )

for( int column = 0; column < imageWidth; column ++ )

{

if( isBottomUp ) // [ 自下而上, 自左而右的图像数据存储顺序 ]

{

swapBuffer[ row \* imageWidth \* 3 + column \* 3 ] = bitmapData[ ( imageHeight - row - 1 ) \* bytesPerFileRow + column \* 4 ];

swapBuffer[ row \* imageWidth \* 3 + column \* 3 + 1 ] = bitmapData[ ( imageHeight - row - 1 ) \* bytesPerFileRow + column \* 4 + 1 ];

swapBuffer[ row \* imageWidth \* 3 + column \* 3 + 2 ] = bitmapData[ ( imageHeight - row - 1 ) \* bytesPerFileRow + column \* 4 + 2 ];

}

else // [ 自上而下, 自左而右的图像数据存储顺序 ]

{

swapBuffer[ row \* imageWidth \* 3 + column \* 3 ] = bitmapData[ row \* bytesPerFileRow + column \* 4 ];

swapBuffer[ row \* imageWidth \* 3 + column \* 3 + 1 ] = bitmapData[ row \* bytesPerFileRow + column \* 4 + 1 ];

swapBuffer[ row \* imageWidth \* 3 + column \* 3 + 2 ] = bitmapData[ row \* bytesPerFileRow + column \* 4 + 2 ];

}

}

m\_colorImage.ImportFrom( swapBuffer ); // [ 导入图像内容 ]

delete [] swapBuffer; // [ 释放缓冲区 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] .................................................

// [ 导入 RAW 图像文件 ] .................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] .................................................

void CImage::Import\_from\_raw\_file( const CString& file\_name )

{

// [ 此 RAW 格式图像导入函数，只处理一种 RAW 格式，即图像像素高度和 ]

// [ 宽度都为 512，并且图像数据是按照“BGR，BGR，...”的顺序进行存 ]

// [ 储。对于其它格式的 RAW 图像文件，则一律放弃数据读取。 ]

const long image\_height = 512; // [ 图像像素高度 ]

const long image\_width = 512; // [ 图像像素宽度 ]

CFile raw\_file( file\_name, CFile::modeRead ); // [ 读打开 RAW 图像文件 ]

if( raw\_file.GetLength() != image\_height \* image\_width \* 3 ) // [ 如果图像尺寸不匹配，那么 ... ]

{

raw\_file.Close(); // [ 关闭 RAW 图像文件 ]

return; // [ 结束返回 ]

}

m\_imageType = Color; // [ 设为彩色图像 ]

m\_colorImage.Construction( image\_height, image\_width ); // [ 构造图像 ]

raw\_file.Read( &( m\_colorImage[0][0] ), m\_colorImage.GetSizeInBytes() ); // [ 读取 RAW 图像文件数据 ]

raw\_file.Close(); // [ 关闭 RAW 图像文件 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...........................................

// [ 导入 24 位 BMP 图像数据 ] ...........................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...........................................

void CImage::ImportFrom\_24\_BmpFile( BYTE\* bitmapData, bool isBottomUp,

long imageHeight, long imageWidth,

long bytesPerMemoryRow, long bytesPerFileRow )

{

m\_imageType = Color; // [ 设为彩色图像 ]

m\_colorImage.Construction( imageHeight, imageWidth ); // [ 构造图像 ]

BYTE\* swapBuffer = new BYTE[ bytesPerMemoryRow \* imageHeight ]; // [ 申请缓冲区 ]

if( isBottomUp ) // [ 自下而上, 自左而右的图像数据存储顺序 ]

for( int row = 0; row < imageHeight; row++ )

memcpy( swapBuffer + row \* bytesPerMemoryRow,

bitmapData + ( imageHeight - row - 1 ) \* bytesPerFileRow,

bytesPerMemoryRow); // [ 图像上下翻转 ]

else // [ 自上而下, 自左而右的图像数据存储顺序 ]

for( int row = 0; row < imageHeight; row++ )

memcpy( swapBuffer + row \* bytesPerMemoryRow,

bitmapData + row \* bytesPerFileRow,

bytesPerMemoryRow);

m\_colorImage.ImportFrom( swapBuffer ); // [ 导入图像内容 ]

delete [] swapBuffer; // [ 释放缓冲区 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ............................................

// [ 导入 8 位 BMP 图像数据 ] ............................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ............................................

void CImage::ImportFrom\_8\_BmpFile( BYTE\* bitmapData, RGBQUAD\* colorTemplate, bool isBottomUp,

long imageHeight, long imageWidth,

long bytesPerMemoryRow, long bytesPerFileRow )

{

m\_imageType = Gray; // [ 设为灰度图像 ]

m\_grayImage.Construction( imageHeight, imageWidth ); // [ 构造图像 ]

BYTE\* swapBuffer = new BYTE[ imageHeight \* imageWidth ]; // [ 申请缓冲区 ]

for( int row = 0; row < imageHeight; row++ )

for( int column = 0; column < imageWidth; column++ )

{

BYTE colorIndex; // [ 调色板索引 ]

if( isBottomUp ) // [ 自下而上, 自左而右的图像数据存储顺序 ]

colorIndex = bitmapData[ ( imageHeight - row - 1 ) \* bytesPerFileRow + column ];

else // [ 自上而下, 自左而右的图像数据存储顺序 ]

colorIndex = bitmapData[ row \* bytesPerFileRow + column ];

RGBQUAD quad = colorTemplate[ colorIndex ]; // [ 查找调色板 ]

BYTE maxComponent = max( max( quad.rgbBlue, quad.rgbGreen ), quad.rgbRed ) ; // [ 最大颜色分量 ]

swapBuffer[ row \* imageWidth + column ] = maxComponent ; // [ 灰度设置 ]

}

m\_grayImage.ImportFrom( swapBuffer ); // [ 导入图像内容 ]

delete [] swapBuffer; // [ 释放缓冲区 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ............................................

// [ 导入 1 位 BMP 图像数据 ] ............................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ............................................

void CImage::ImportFrom\_1\_BmpFile( BYTE\* bitmapData, RGBQUAD\* colorTemplate, bool isBottomUp,

long imageHeight, long imageWidth,

long bytesPerMemoryRow, long bytesPerFileRow )

{

m\_imageType = Unicolor; // [ 设为二值图像 ]

m\_unicolorImage.Construction( imageHeight, imageWidth ); // [ 构造图像 ]

BlackWhite\* swapBuffer = new BlackWhite[ imageWidth \* imageHeight ] ; // [ 申请缓冲区 ]

for( int row = 0; row < imageHeight; row++ )

for( int column = 0; column < imageWidth; column++ )

{

long indexByte = column / 8 ; // [ 像素所在字节索引 ]

long indexBit = 7 - column % 8 ; // [ 像素所在位索引( 相对最后一位 ) ]

BYTE entireByte; // [ 像素所在字节 ]

if( isBottomUp ) // [ 自下而上, 自左而右的图像数据存储顺序 ]

entireByte = bitmapData[ ( imageHeight - row - 1 ) \* bytesPerFileRow + indexByte ];

else // [ 自上而下, 自左而右的图像数据存储顺序 ]

entireByte = bitmapData[ row \* bytesPerFileRow + indexByte ];

BYTE colorIndex = ( entireByte >> indexBit ) & 1 ; // [ 调色板索引 ]

RGBQUAD quad = colorTemplate[ colorIndex ]; // [ 查找调色板 ]

BYTE maxComponent = max( max( quad.rgbBlue, quad.rgbGreen ), quad.rgbRed ) ; // [ 最大颜色分量 ]

swapBuffer[ row \* imageWidth + column ] = ( maxComponent > 128 ) ? White : Black ; // [ 二值转换 ]

}

m\_unicolorImage.ImportFrom( swapBuffer ); // [ 导入图像内容 ]

delete [] swapBuffer; // [ 释放缓冲区 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 显示彩色图像 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

void CImage::ShowImage( const CTMatrix< RGB\_TRIPLE >& colorImage, CDC\* pDC, CPoint leftTop, CSize displaySize )

{

if( colorImage.IsNull() ) // [ 判断图像是否为空 ]

return;

long imageHeight = colorImage.GetRows( ); // [ 图像高度( 像素 ) ]

long imageWidth = colorImage.GetColumns( ); // [ 图像宽度( 像素 ) ]

long bytesPerMemoryRow = imageWidth \* sizeof( RGB\_TRIPLE ); // [ 内存中图像行存储字节数 ]

long bytesPerFileRow = ( ( bytesPerMemoryRow + 3 ) / 4 ) \* 4 ; // [ 文件中图像行存储字节数 ]

BITMAPINFO\* bitmapInfo = new BITMAPINFO ; // [ 申请位图信息内存 ]

BYTE\* bitmapData = new BYTE[ colorImage.GetSizeInBytes() ]; // [ 申请位图数据内存 ]

BYTE\* swapBuffer = ( bytesPerMemoryRow == bytesPerFileRow ) ?

bitmapData : ( new BYTE[ imageHeight \* bytesPerFileRow ] ); // [ 申请图像显示缓冲区 ]

Construct\_24\_Info( bitmapInfo, imageHeight, imageWidth ); // [ 构造 24 位信息 ]

colorImage.ExportTo( bitmapData ); // [ 导出图像内容 ]

if( bytesPerMemoryRow != bytesPerFileRow )

for( int row = 0; row < imageHeight; row++ )

memcpy( swapBuffer + row \* bytesPerFileRow,

bitmapData + row \* bytesPerMemoryRow,

bytesPerMemoryRow ); // [ 设置图像显示缓冲区内容 ]

if( displaySize == CSize(0, 0) ) // [ 正常显示 ]

{

::SetDIBitsToDevice( pDC->GetSafeHdc(), leftTop.x, leftTop.y,

(DWORD)imageWidth, (DWORD)imageHeight,

0, 0, 0, (UINT)imageHeight,

swapBuffer, bitmapInfo,

DIB\_RGB\_COLORS);

}

else // [ 伸缩显示 ]

{

int oldMode = pDC->GetStretchBltMode( ); // [ 保存旧的位图伸缩模式 ]

pDC->SetStretchBltMode( COLORONCOLOR ); // [ 设置新的位图伸缩模式 ]

::StretchDIBits( pDC->GetSafeHdc(), leftTop.x, leftTop.y,

displaySize.cx, displaySize.cy,

0, 0, imageWidth, imageHeight,

swapBuffer, bitmapInfo,

DIB\_RGB\_COLORS, SRCCOPY);

pDC->SetStretchBltMode( oldMode ); // [ 恢复旧的位图伸缩模式 ]

}

if( bytesPerMemoryRow != bytesPerFileRow ) delete [] swapBuffer; // [ 释放图像显示缓冲区 ]

delete [] bitmapData; // [ 释放位图数据内存 ]

delete bitmapInfo; // [ 释放位图信息内存 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 显示灰度图像 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

void CImage::ShowImage( const CTMatrix< BYTE >& grayImage, CDC\* pDC, CPoint leftTop, CSize displaySize )

{

if( grayImage.IsNull() ) // [ 判断图像是否为空 ]

return;

long imageHeight = grayImage.GetRows(); // [ 图像高度( 像素 ) ]

long imageWidth = grayImage.GetColumns(); // [ 图像宽度( 像素 ) ]

long bytesPerFileRow = ( ( imageWidth + 3 ) / 4 ) \* 4 ; // [ 文件中图像行存储字节数 ]

BITMAPINFO\* bitmapInfo = ( BITMAPINFO\* )( new BYTE[ sizeof( BITMAPINFOHEADER ) + 256 \* sizeof( RGBQUAD ) ] ); // [ 申请位图信息内存 ]

BYTE\* bitmapData = new BYTE[ grayImage.GetSizeInBytes() ]; // [ 申请位图数据内存 ]

BYTE\* swapBuffer = ( imageWidth == bytesPerFileRow ) ?

bitmapData : ( new BYTE[ imageHeight \* bytesPerFileRow ] ); // [ 申请图像显示缓冲区 ]

Construct\_8\_Info( bitmapInfo, imageHeight, imageWidth ); // [ 构造 8 位信息 ]

grayImage.ExportTo( bitmapData ); // [ 导出图像内容 ]

if( imageWidth != bytesPerFileRow )

for( int row = 0; row < imageHeight; row++ )

memcpy( swapBuffer + row \* bytesPerFileRow,

bitmapData + row \* imageWidth,

imageWidth ); // [ 设置图像显示缓冲区内容 ]

if( displaySize == CSize(0, 0) ) // [ 正常显示 ]

{

::SetDIBitsToDevice( pDC->GetSafeHdc(), leftTop.x, leftTop.y,

(DWORD)imageWidth, (DWORD)imageHeight,

0, 0, 0, (UINT)imageHeight,

swapBuffer, bitmapInfo,

DIB\_RGB\_COLORS);

}

else // [ 伸缩显示 ]

{

int oldMode = pDC->GetStretchBltMode( ); // [ 保存旧的位图伸缩模式 ]

pDC->SetStretchBltMode( COLORONCOLOR ); // [ 设置新的位图伸缩模式 ]

::StretchDIBits( pDC->GetSafeHdc(), leftTop.x, leftTop.y,

displaySize.cx, displaySize.cy,

0, 0, imageWidth, imageHeight,

swapBuffer, bitmapInfo,

DIB\_RGB\_COLORS, SRCCOPY);

pDC->SetStretchBltMode( oldMode ); // [ 恢复旧的位图伸缩模式 ]

}

if( imageWidth != bytesPerFileRow ) delete [] swapBuffer; // [ 释放图像显示缓冲区 ]

delete [] bitmapData; // [ 释放位图数据内存 ]

delete [] bitmapInfo; // [ 释放位图信息内存 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 显示二值图像 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

void CImage::ShowImage( const CTMatrix< BlackWhite >& unicolorImage, CDC\* pDC, CPoint leftTop, CSize displaySize )

{

if( unicolorImage.IsNull() ) // [ 判断图像是否为空 ]

return;

long imageHeight = unicolorImage.GetRows(); // [ 图像高度( 像素 ) ]

long imageWidth = unicolorImage.GetColumns(); // [ 图像宽度( 像素 ) ]

BITMAPINFO\* bitmapInfo = new BITMAPINFO ; // [ 申请位图信息内存 ]

BYTE\* bitmapData = new BYTE[ unicolorImage.GetSizeInBytes() ]; // [ 申请位图数据内存 ]

Construct\_32\_Info( bitmapInfo, imageHeight, imageWidth ); // [ 构造 32 信息 ]

unicolorImage.ExportTo( bitmapData ); // [ 导出图像内容 ]

if( displaySize == CSize(0, 0) ) // [ 正常显示 ]

{

::SetDIBitsToDevice( pDC->GetSafeHdc(), leftTop.x, leftTop.y,

(DWORD)imageWidth, (DWORD)imageHeight,

0, 0, 0, (UINT)imageHeight,

bitmapData, bitmapInfo,

DIB\_RGB\_COLORS);

}

else // [ 伸缩显示 ]

{

int oldMode = pDC->GetStretchBltMode( ); // [ 保存旧的位图伸缩模式 ]

pDC->SetStretchBltMode( COLORONCOLOR ); // [ 设置新的位图伸缩模式 ]

::StretchDIBits( pDC->GetSafeHdc(), leftTop.x, leftTop.y,

displaySize.cx, displaySize.cy,

0, 0, imageWidth, imageHeight,

bitmapData, bitmapInfo,

DIB\_RGB\_COLORS, SRCCOPY);

pDC->SetStretchBltMode( oldMode ); // [ 恢复旧的位图伸缩模式 ]

}

delete [] bitmapData; // [ 释放位图数据内存 ]

delete bitmapInfo; // [ 释放位图信息内存 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\* ] .....................................................

// [ 构造 1 位信息 ] .....................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\* ] .....................................................

void CImage::Construct\_1\_Info( BITMAPINFO\* bitmapInfo, long imageHeight, long imageWidth )

{

long bytesPerMemoryRow = ( imageWidth + 7 ) / 8 ; // [ 内存中图像行存储字节数 ]

long bytesPerFileRow = ( ( bytesPerMemoryRow + 3 ) / 4 ) \* 4 ; // [ 文件中图像行存储字节数 ]

bitmapInfo->bmiHeader.biBitCount = 1 ;

bitmapInfo->bmiHeader.biClrImportant = 0;

bitmapInfo->bmiHeader.biClrUsed = 0;

bitmapInfo->bmiHeader.biCompression = BI\_RGB;

bitmapInfo->bmiHeader.biHeight = - imageHeight;

bitmapInfo->bmiHeader.biPlanes = 1;

bitmapInfo->bmiHeader.biSizeImage = imageHeight \* bytesPerFileRow ;

bitmapInfo->bmiHeader.biSize = 40;

bitmapInfo->bmiHeader.biWidth = imageWidth;

bitmapInfo->bmiHeader.biXPelsPerMeter = 3780;

bitmapInfo->bmiHeader.biYPelsPerMeter = 3780;

RGBQUAD\* colorTemplate = bitmapInfo->bmiColors;

colorTemplate[0].rgbBlue = BYTE( 0 );

colorTemplate[0].rgbGreen = BYTE( 0 );

colorTemplate[0].rgbRed = BYTE( 0 );

colorTemplate[0].rgbReserved = BYTE( 0 );

colorTemplate[1].rgbBlue = BYTE( 255 );

colorTemplate[1].rgbGreen = BYTE( 255 );

colorTemplate[1].rgbRed = BYTE( 255 );

colorTemplate[1].rgbReserved = BYTE( 0 );

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...................................................

// [ 构造 1 位文件头 ] ...................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...................................................

void CImage::Construct\_1\_FileHeader( BITMAPFILEHEADER\* fileHeader, long imageHeight, long imageWidth )

{

long bytesPerMemoryRow = ( imageWidth + 7 ) / 8 ; // [ 内存中图像行存储字节数 ]

long bytesPerFileRow = ( ( bytesPerMemoryRow + 3 ) / 4 ) \* 4 ; // [ 文件中图像行存储字节数 ]

fileHeader->bfType = ((WORD) ('M' << 8) | 'B') ;

fileHeader->bfReserved1 = 0 ;

fileHeader->bfReserved2 = 0 ;

fileHeader->bfOffBits = sizeof( BITMAPFILEHEADER ) + sizeof( BITMAPINFOHEADER ) + 2 \* sizeof( RGBQUAD ) ;

fileHeader->bfSize = fileHeader->bfOffBits + imageHeight \* bytesPerFileRow ;

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\* ] .....................................................

// [ 构造 8 位信息 ] .....................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\* ] .....................................................

void CImage::Construct\_8\_Info( BITMAPINFO\* bitmapInfo, long imageHeight, long imageWidth )

{

bitmapInfo->bmiHeader.biBitCount = 8 ;

bitmapInfo->bmiHeader.biClrImportant = 0;

bitmapInfo->bmiHeader.biClrUsed = 0;

bitmapInfo->bmiHeader.biCompression = BI\_RGB;

bitmapInfo->bmiHeader.biHeight = - imageHeight;

bitmapInfo->bmiHeader.biPlanes = 1;

bitmapInfo->bmiHeader.biSizeImage = imageHeight \* ( ( ( imageWidth + 3 ) / 4 ) \* 4 ) ;

bitmapInfo->bmiHeader.biSize = 40;

bitmapInfo->bmiHeader.biWidth = imageWidth;

bitmapInfo->bmiHeader.biXPelsPerMeter = 3780;

bitmapInfo->bmiHeader.biYPelsPerMeter = 3780;

RGBQUAD\* colorTemplate = bitmapInfo->bmiColors;

for( int index = 0; index < 256; index++ )

{

colorTemplate[index].rgbBlue = BYTE( index );

colorTemplate[index].rgbGreen = BYTE( index );

colorTemplate[index].rgbRed = BYTE( index );

colorTemplate[index].rgbReserved = BYTE( 0 );

}

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...................................................

// [ 构造 8 位文件头 ] ...................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...................................................

void CImage::Construct\_8\_FileHeader( BITMAPFILEHEADER\* fileHeader, long imageHeight, long imageWidth )

{

fileHeader->bfType = ((WORD) ('M' << 8) | 'B') ;

fileHeader->bfReserved1 = 0 ;

fileHeader->bfReserved2 = 0 ;

fileHeader->bfOffBits = sizeof( BITMAPFILEHEADER ) + sizeof( BITMAPINFOHEADER ) + 256 \* sizeof( RGBQUAD ) ;

fileHeader->bfSize = fileHeader->bfOffBits + imageHeight \* ( ( ( imageWidth + 3 ) / 4 ) \* 4 ) ;

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

// [ 构造 24 位信息 ] ....................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

void CImage::Construct\_24\_Info( BITMAPINFO\* bitmapInfo, long imageHeight, long imageWidth )

{

bitmapInfo->bmiHeader.biBitCount = 24;

bitmapInfo->bmiHeader.biClrImportant = 0;

bitmapInfo->bmiHeader.biClrUsed = 0;

bitmapInfo->bmiHeader.biCompression = BI\_RGB;

bitmapInfo->bmiHeader.biHeight = - imageHeight;

bitmapInfo->bmiHeader.biPlanes = 1;

bitmapInfo->bmiHeader.biSizeImage = imageHeight \* ( ( ( ( imageWidth \* 3 ) + 3 ) / 4 ) \* 4 );

bitmapInfo->bmiHeader.biSize = 40;

bitmapInfo->bmiHeader.biWidth = imageWidth;

bitmapInfo->bmiHeader.biXPelsPerMeter = 3780;

bitmapInfo->bmiHeader.biYPelsPerMeter = 3780;

bitmapInfo->bmiColors[0].rgbBlue = (BYTE)0;

bitmapInfo->bmiColors[0].rgbGreen = (BYTE)0;

bitmapInfo->bmiColors[0].rgbRed = (BYTE)0;

bitmapInfo->bmiColors[0].rgbReserved = (BYTE)0;

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

// [ 构造 24 位文件头 ] ..................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ..................................................

void CImage::Construct\_24\_FileHeader( BITMAPFILEHEADER\* fileHeader, long imageHeight, long imageWidth )

{

fileHeader->bfType = ((WORD) ('M' << 8) | 'B') ;

fileHeader->bfReserved1 = 0 ;

fileHeader->bfReserved2 = 0 ;

fileHeader->bfOffBits = 54 ;

fileHeader->bfSize = fileHeader->bfOffBits + imageHeight \* ( ( ( ( imageWidth \* 3 ) + 3 ) / 4 ) \* 4 ) ;

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

// [ 构造 32 位信息 ] ....................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ....................................................

void CImage::Construct\_32\_Info( BITMAPINFO\* bitmapInfo, long imageHeight, long imageWidth )

{

bitmapInfo->bmiHeader.biBitCount = 32;

bitmapInfo->bmiHeader.biClrImportant = 0;

bitmapInfo->bmiHeader.biClrUsed = 0;

bitmapInfo->bmiHeader.biCompression = BI\_RGB;

bitmapInfo->bmiHeader.biHeight = - imageHeight;

bitmapInfo->bmiHeader.biPlanes = 1;

bitmapInfo->bmiHeader.biSizeImage = imageHeight \* imageWidth \* 4 ;

bitmapInfo->bmiHeader.biSize = 40;

bitmapInfo->bmiHeader.biWidth = imageWidth;

bitmapInfo->bmiHeader.biXPelsPerMeter = 3780;

bitmapInfo->bmiHeader.biYPelsPerMeter = 3780;

bitmapInfo->bmiColors[0].rgbBlue = (BYTE)0;

bitmapInfo->bmiColors[0].rgbGreen = (BYTE)0;

bitmapInfo->bmiColors[0].rgbRed = (BYTE)0;

bitmapInfo->bmiColors[0].rgbReserved = (BYTE)0;

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 保存彩色图像 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

void CImage::SaveImage( const CTMatrix< RGB\_TRIPLE >& colorImage, const CString& fileName, int jpeg\_quality )

{

CString postfix = PostfixToFileName( fileName ); // [ 获取文件名后缀 ]

if( postfix.CompareNoCase( "BMP" ) == 0 ) // [ BMP 图像 ]

{

SaveImageToBmpFile( colorImage, fileName ); // [ 保存彩色图像为 BMP 文件 ]

return; // [ 结束返回 ]

}

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 保存灰度图像 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

void CImage::SaveImage( const CTMatrix< BYTE >& grayImage, const CString& fileName, int jpeg\_quality )

{

CString postfix = PostfixToFileName( fileName ); // [ 获取文件名后缀 ]

if( postfix.CompareNoCase( "BMP" ) == 0 ) // [ BMP 图像 ]

{

SaveImageToBmpFile( grayImage, fileName ); // [ 保存灰度图像为 BMP 文件 ]

return; // [ 结束返回 ]

}

}

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

// [ 保存二值图像 ] ......................................................

// [ \*\*\*\*\*\*\*\*\*\*\*\* ] ......................................................

void CImage::SaveImage( const CTMatrix< BlackWhite >& unicolorImage, const CString& fileName )

{

CString postfix = PostfixToFileName( fileName ); // [ 获取文件名后缀 ]

if( postfix.CompareNoCase( "BMP" ) == 0 ) // [ BMP 图像 ]

{

SaveImageToBmpFile( unicolorImage, fileName ); // [ 保存二值图像为 BMP 文件 ]

return; // [ 结束返回 ]

}

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...........................................

// [ 保存彩色图像为 BMP 文件 ] ...........................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...........................................

void CImage::SaveImageToBmpFile( const CTMatrix< RGB\_TRIPLE >& colorImage, const CString& fileName )

{

if( colorImage.IsNull() ) // [ 判断图像是否为空 ]

return;

CFile bitmapFile; // [ Bmp 文件 ]

if( !bitmapFile.Open( fileName, CFile::modeCreate | CFile::modeWrite ) ) // [ 打开文件 ]

return;

long imageHeight = colorImage.GetRows( ); // [ 图像高度( 像素 ) ]

long imageWidth = colorImage.GetColumns( ); // [ 图像宽度( 像素 ) ]

long bytesPerMemoryRow = imageWidth \* sizeof( RGB\_TRIPLE ); // [ 内存中图像行存储字节数 ]

long bytesPerFileRow = ( ( bytesPerMemoryRow + 3 ) / 4 ) \* 4 ; // [ 文件中图像行存储字节数 ]

BITMAPFILEHEADER\* fileHeader = new BITMAPFILEHEADER ; // [ 申请文件头内存 ]

BITMAPINFO\* bitmapInfo = new BITMAPINFO ; // [ 申请位图信息内存 ]

BYTE\* bitmapData = new BYTE[ colorImage.GetSizeInBytes() ]; // [ 申请位图数据内存 ]

BYTE\* swapBuffer = ( bytesPerMemoryRow == bytesPerFileRow ) ?

bitmapData : ( new BYTE[ imageHeight \* bytesPerFileRow ] ); // [ 申请图像保存缓冲区 ]

Construct\_24\_FileHeader( fileHeader, imageHeight, imageWidth ); // [ 构造 24 位文件头 ]

Construct\_24\_Info( bitmapInfo, imageHeight, imageWidth ); // [ 构造 24 位信息 ]

colorImage.ExportTo( bitmapData ); // [ 导出图像内容 ]

if( bytesPerMemoryRow != bytesPerFileRow )

for( int row = 0; row < imageHeight; row++ )

memcpy( swapBuffer + row \* bytesPerFileRow,

bitmapData + row \* bytesPerMemoryRow,

bytesPerMemoryRow ); // [ 设置图像保存缓冲区内容 ]

bitmapFile.Write( fileHeader, sizeof( BITMAPFILEHEADER ) ); // [ 保存文件头 ]

bitmapFile.Write( bitmapInfo, sizeof( BITMAPINFOHEADER ) ); // [ 保存信息头 ]

bitmapFile.Write( swapBuffer, bytesPerFileRow \* imageHeight ); // [ 保存图像内容 ]

if( bytesPerMemoryRow != bytesPerFileRow ) delete [] swapBuffer; // [ 释放图像保存缓冲区 ]

delete [] bitmapData; // [ 释放位图数据内存 ]

delete bitmapInfo; // [ 释放位图信息内存 ]

delete fileHeader; // [ 释放文件头内存 ]

bitmapFile.Close(); // [ 关闭文件 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...........................................

// [ 保存灰度图像为 BMP 文件 ] ...........................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...........................................

void CImage::SaveImageToBmpFile( const CTMatrix< BYTE >& grayImage, const CString& fileName )

{

if( grayImage.IsNull() ) // [ 判断图像是否为空 ]

return;

CFile bitmapFile; // [ Bmp 文件 ]

if( !bitmapFile.Open( fileName, CFile::modeCreate | CFile::modeWrite ) ) // [ 打开文件 ]

return;

long imageHeight = grayImage.GetRows(); // [ 图像高度( 像素 ) ]

long imageWidth = grayImage.GetColumns(); // [ 图像宽度( 像素 ) ]

long bytesPerFileRow = ( ( imageWidth + 3 ) / 4 ) \* 4 ; // [ 文件中图像行存储字节数 ]

BITMAPFILEHEADER\* fileHeader = new BITMAPFILEHEADER ; // [ 申请文件头内存 ]

BITMAPINFO\* bitmapInfo = ( BITMAPINFO\* )( new BYTE[ sizeof( BITMAPINFOHEADER ) + 256 \* sizeof( RGBQUAD ) ] ); // [ 申请位图信息内存 ]

BYTE\* bitmapData = new BYTE[ grayImage.GetSizeInBytes() ]; // [ 申请位图数据内存 ]

BYTE\* swapBuffer = ( imageWidth == bytesPerFileRow ) ?

bitmapData : ( new BYTE[ imageHeight \* bytesPerFileRow ] ); // [ 申请图像保存缓冲区 ]

Construct\_8\_FileHeader( fileHeader, imageHeight, imageWidth ); // [ 构造 8 位文件头 ]

Construct\_8\_Info( bitmapInfo, imageHeight, imageWidth ); // [ 构造 8 位信息 ]

grayImage.ExportTo( bitmapData ); // [ 导出图像内容 ]

if( imageWidth != bytesPerFileRow )

for( int row = 0; row < imageHeight; row++ )

memcpy( swapBuffer + row \* bytesPerFileRow,

bitmapData + row \* imageWidth,

imageWidth ); // [ 设置图像保存缓冲区内容 ]

bitmapFile.Write( fileHeader, sizeof( BITMAPFILEHEADER ) ); // [ 保存文件头 ]

bitmapFile.Write( bitmapInfo, sizeof( BITMAPINFOHEADER ) + 256 \* sizeof( RGBQUAD ) ); // [ 保存信息头 ]

bitmapFile.Write( swapBuffer, bytesPerFileRow \* imageHeight ); // [ 保存图像内容 ]

if( imageWidth != bytesPerFileRow ) delete [] swapBuffer; // [ 释放图像保存缓冲区 ]

delete [] bitmapData; // [ 释放位图数据内存 ]

delete [] bitmapInfo; // [ 释放位图信息内存 ]

delete fileHeader; // [ 释放文件头内存 ]

bitmapFile.Close(); // [ 关闭文件 ]

}

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...........................................

// [ 保存二值图像为 BMP 文件 ] ...........................................

// [ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ] ...........................................

void CImage::SaveImageToBmpFile( const CTMatrix< BlackWhite >& unicolorImage, const CString& fileName )

{

if( unicolorImage.IsNull() ) // [ 判断图像是否为空 ]

return;

CFile bitmapFile; // [ Bmp 文件 ]

if( !bitmapFile.Open( fileName, CFile::modeCreate | CFile::modeWrite ) ) // [ 打开文件 ]

return;

long imageHeight = unicolorImage.GetRows(); // [ 图像高度( 像素 ) ]

long imageWidth = unicolorImage.GetColumns(); // [ 图像宽度( 像素 ) ]

long bytesPerMemoryRow = ( imageWidth + 7 ) / 8 ; // [ 内存中图像行存储字节数 ]

long bytesPerFileRow = ( ( bytesPerMemoryRow + 3 ) / 4 ) \* 4 ; // [ 文件中图像行存储字节数 ]

BITMAPFILEHEADER\* fileHeader = new BITMAPFILEHEADER ; // [ 申请文件头内存 ]

BITMAPINFO\* bitmapInfo = ( BITMAPINFO\* )( new BYTE[ sizeof( BITMAPINFOHEADER ) + 2 \* sizeof( RGBQUAD ) ] ); // [ 申请位图信息内存 ]

BYTE\* swapBuffer = new BYTE[ imageHeight \* bytesPerFileRow ]; // [ 申请图像保存缓冲区 ]

Construct\_1\_FileHeader( fileHeader, imageHeight, imageWidth ); // [ 构造 1 位文件头 ]

Construct\_1\_Info( bitmapInfo, imageHeight, imageWidth ); // [ 构造 1 位信息 ]

for( int row = 0; row < imageHeight; row++ )

for( int column = 0; column < imageWidth; column++ )

{

long indexByte = column / 8 ; // [ 像素所在字节索引 ]

long indexBit = 7 - column % 8 ; // [ 像素所在位索引( 相对最后一位 ) ]

if( unicolorImage[row][column] == White )

swapBuffer[ row \* bytesPerFileRow + indexByte ] |= ( 1 << indexBit ); // [ 设置 1 ]

else

swapBuffer[ row \* bytesPerFileRow + indexByte ] &= ( 1 << indexBit ) ^ 0xFF; // [ 设置 0 ]

}

bitmapFile.Write( fileHeader, sizeof( BITMAPFILEHEADER ) ); // [ 保存文件头 ]

bitmapFile.Write( bitmapInfo, sizeof( BITMAPINFOHEADER ) + 2 \* sizeof( RGBQUAD ) ); // [ 保存信息头 ]

bitmapFile.Write( swapBuffer, bytesPerFileRow \* imageHeight ); // [ 保存图像内容 ]

delete [] swapBuffer; // [ 释放图像保存缓冲区 ]

delete [] bitmapInfo; // [ 释放位图信息内存 ]

delete fileHeader; // [ 释放文件头内存 ]

bitmapFile.Close(); // [ 关闭文件 ]

}