# 数字媒体处理技术实验

# ——基于MFC/OPENCV的可视化程序开发与图像处理算法

[数字媒体处理技术实验 1](#_Toc10690)

[——基于MFC/OPENCV的可视化程序开发与图像处理算法 1](#_Toc4359)

[一 opencv配置 1](#_Toc27517)

[二 MFC 窗口程序设计 4](#_Toc11196)

[Windows应用程序结构 4](#_Toc15055)

[消息与消息映射 6](#_Toc8724)

[MFC文档/视图框架中各个对象的关系 9](#_Toc25336)

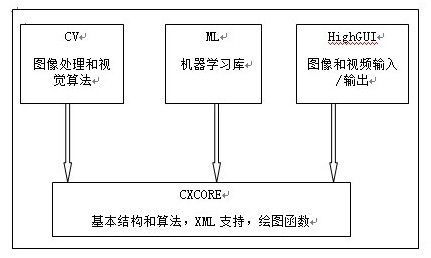
[建立窗口程序示例 10](#_Toc24484)

[三 多线程图像处理程序设计 15](#_Toc325)

# 一 opencv配置

OpenCV是一个开源的计算机视觉库，其全称为：Open Source Computer Vision Library。它于1999年由Intel建立，现在由Willow Garage提供支持。其是一个基于BSD许可证授权（开源）发行的跨平台计算机视觉库，可以运行在Linux、Windows和Mac OS操作系统上。OpenCV的一个目标是构建一个简单易用的计算机视觉框架，以帮助开发人员更快速便捷地设计更复杂的计算机视觉相关的应用程序。其包含500多个函数，覆盖了计算机视觉的许多应用领域，比如用户界面、信息安全、立体视觉、产品检测、机器学习等。

OpenCV主体分为五个模块，其中的四个模块如图所示。OpenCV的CV模块包含基本的图像处理函数和高级的计算机视觉算法。ML为机器学习库，一些基于统计的分类和聚类工具包含于其中。HighGUI包含图像和视频输入/输出的函数。CXCore包含一些基本的数据结构和相关的函数。图3-3中并没有包含CvAux模块，该模块中一般存放一些即将淘汰的算法和函数，同时还有一些新出现的实验性的算法和函数。



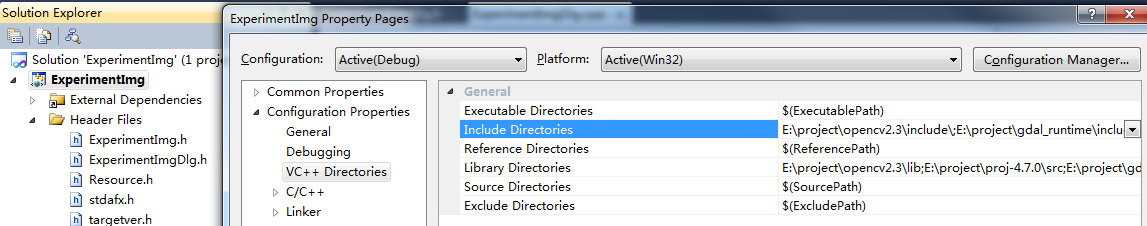
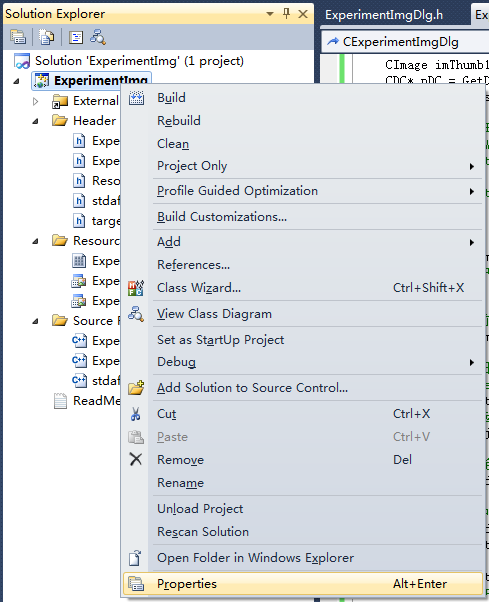
OpenCV的基本结构

OpenCV也提供了对MFC的支持，CvvImage类里面提供了Show()和DrawToHDC()两个方法将图像显示到对指定的区域。

1. 安装c:\opencv
2. 这让系统在运行opencv程序的时候知道去哪里找dll文件，以下皆可：
3. 添加环境变量OPENCV：c:\openc

添加path：c:\opencv\build\x86\vc10\bin。

1. 将对应dll拷贝到c:\windows\system32目录下。
2. 将dll拷贝到debug或release等exe输出目录下。
3. 配置VC环境



修改vc++ directories 下面的include directories  和 library directories 值。

   include directories的值在原值后面加上

   ;$(OPENCV)\build\include

   ;$(OPENCV)\build\include\opencv

   ;$(OPENCV)\build\include\opencv2

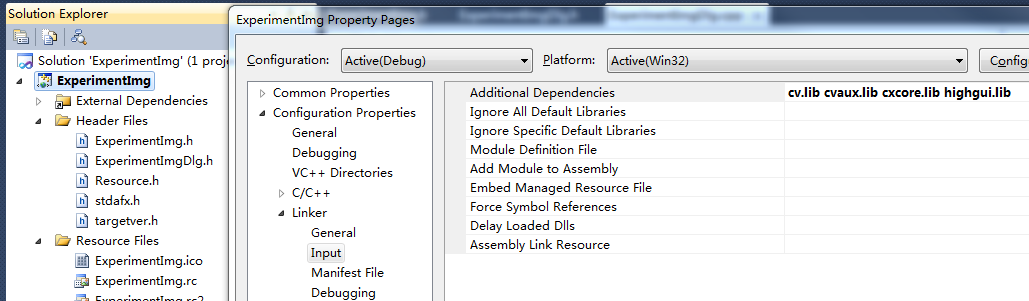
   library directories的值在原值后面加上

   ;$(OPENCV)\build\x86\vc10\lib

注意：

1这里不要同时加上;$(OPENCV)\build\x86\vc10\staticlib;不然会报错的。

2 不成功的话可能会报 缺少tbb\_debug.dll,从%OPENCV%\build\common\tbb\ia32\vc10目录拷贝tbb\_debug.dll到目录%OPENCV%\build\x86\vc10\bin下。



在linker下面加入lib依赖项，通常只需要cv.lib cvaux.lib cxcore.lib highgui.lib几个即可

**实验任务1 编程环境与基础**：

查看实验室机器的opencv配置环境；在自己机器上配置、安装opencv

了解VC项目管理配置方式

自己进一步学习基于cmake的项目配置与编译方式

# 二 MFC 窗口程序设计

## Windows应用程序结构

Windows操作系统中，显示给用户的界面是窗体，并使用句柄标识不同的窗体，所谓句柄就是一些复杂对象的标识，比如窗口、按钮、滚动条、光标、图形设备上下文等。然后通过事件和消息窗体命令，并且使用消息队列按照消息发送的先后顺序处理。

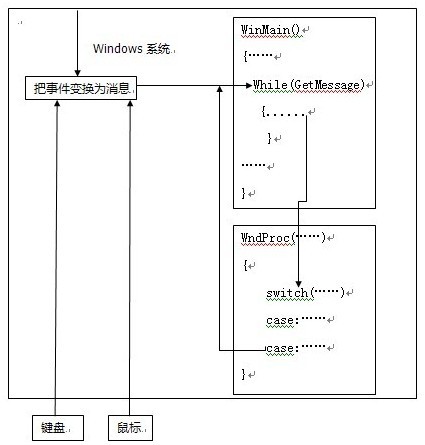


图2-1 Windows系统、应用程序与窗口函数之间的关系

入口函数是所有的Windows程序都必须拥有的，它作为一个程序开始执行的切入点。在默认的情况下，以WinMain()作为应用程序的入口。在Win32平台下，它的函数声明如下：

int APIENTRY WinMain(

HINSTANCE hInstance, //当前实例句柄

HINSTANCE hPrevInstance, //前一个实例句柄

LPSTR lpCmdLine, //序命令行的指针

int nCmdShow //窗口的显示方式标志

)

在我们创建窗口之前，需要调用窗口的注册函数RegisterClass(WINDCLASS &wc)把定义好的窗口样式注册到系统中。一旦使用RegisterClass(WINDCLASS &wc)注册成功，系统中就会存在本次定义的窗口类型，就可以使用这个窗口类型作为一个模板来创建一个或者多个窗口。

然后使用CreateWindow()函数创建窗口。窗口创建成功之后，会返回窗口的句柄。以后就可以用这个句柄类对窗口进行操作。

接着就可以调用ShowWindow()和UpdateWindow()将窗口显示在计算机显示器的屏幕上。窗口完成创建和成功显示后，应用程序的初始化工作也就基本的结束，然后就是重要的消息循环部分。

while(GetMessage(&msg,NULL,NULL,NULL))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

这里GetMessage()是将消息从消息队列中取出来，放到msg变量里面，用户给出的键盘消息是通过TranslateMessage()将其转换为字符消息，然后使用DispatchMessage()函数向系统派送消息。

在Windows的窗口应用程序里面，是通过“窗口函数”来处理各种的消息。在这里开发者需要编写大量的码去处理各种任务。该函数的定义如下：

LRESULT CALLBACK WndProc(

HWND hwnd, //派送消息的窗口句柄

UINT message, //系统传递来的消息标识

WPARAM wParam, //消息的附加参数

LPARAM lParam) //消息的附件参数

图2-1给出了Windows系统，应用程序，窗口函数之间的关系。一个Windows程序员如果想要顺利的开发出一款优秀的Windows应用程序，就非常有必要去深入地理解Windows系统、主函数、窗口函数之间的相互关系。这里Windows系统负责主函数和窗口函数的调用，应用程序启动以后系统会首先调用主函数。当系统得到了由主函数获取的窗口函数消息后，会根据产生事件的窗口类型提供的函数指针去调用对应的窗口函数，完成事件的响应和消息处理。

## 消息与消息映射

由于Windows应用程序是基于事件驱动的，并且是由系统去接收输入再转给应用程序。应用程序如果想要去调用某个函数获取用户的输入，就需要等待系统去接收输入然后传递给应用程序，应用程序才能去调用对应的函数。前一节讲过，每一个窗口都有其对应的窗口函数。应用程序运行过程中，当有对应的窗口输入时，系统就会调用与之对应的窗口函数，处理输入和执行对系统的控制。

在Windows应用程序中，输入是通过消息的形式传递给窗口函数的。当系统接受到一个输入事件，就会在系统中产生一条对应的消息。消息可以通过系统或者应用程序去创建，由此可将消息分为系统消息和应用程序消息。也可以根据消息的发送途径，将消息划分为队列消息和非队列消息。队列消息是指发送消息到先进先出的系统消息队列中，然后到线程消息队列。队列消息用于存储用户通过鼠标或键盘给出的输入，也可以存储重绘消息和退出消息等。在程序中要想将消息发送到消息队列中，就要调用PostMessage()函数去完成。非队列消息是使用系统定义内存对象临时存储消息，这种消息是直接的发送给目的窗口的窗口函数。除了第一种队列消息外，其他消息一般都采用这种处理方式。非队列消息是使用SendMessage()函数将消息直接发送给对应窗口的窗口函数。图2-2展示了事件和消息的传递过程。下面为消息的定义结构：

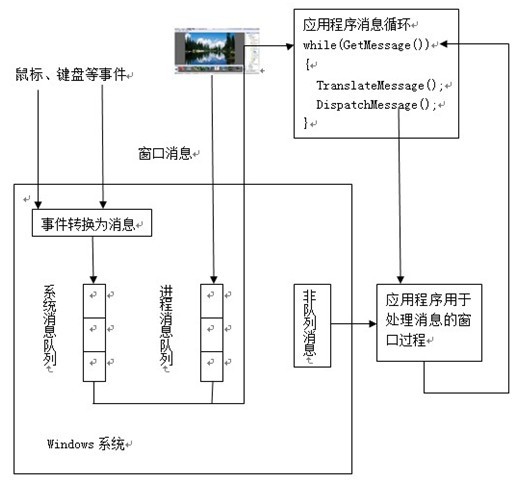


图2-2事件及消息的传递过程

Typedef struct tagMSG

{

HWND hwnd; //产生消息的窗口句柄

UINT message; //消息标识码

WPARAM wParam; //消息附加信息1

LPARAM lParam; //消息/附加信息2

DWORD time; //消息进入消息队列的时刻

POINT pt; //表示发送该消息是光标的位置

}

在传统的Windows应用程序中，窗口函数里面是使用switch-case结构来处理发送给窗口的消息。后来MFC运用了消息映射机制来进行消息的处理。首先，在一个需要进行消息处理的类里面，为每一个消息声明一个消息处理函数，其格式为：afx\_msg void 消息处理函数的名称（）。在每一需要进行消息处理的类里面都会有一张消息映射表，它的表项结构为AFX\_MSGMAP\_ENTRY，其结构如下所示：

Struct AFX\_MSGMAP\_ENTRY

{ UINT nMessage; // Windows消息标识

UINT nCode;

UINT nID;

UINT nLastID;

UINT nSig;

AFX\_PMSG pfn; //消息响应函数的指针

}

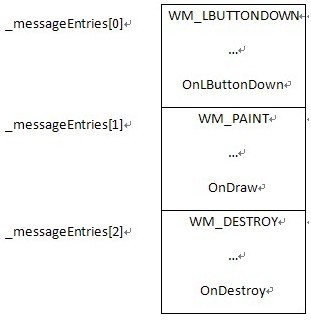


图2-3 类的消息映射表

在需要消息响应的类里面，会在里面声明一个AFX\_MSGMAP\_ENTRY类型的数组\_messageEntries[]，这个数组就作为这个类的消息映射表。图2-3展示了一个类的消息映射表。

MFC使用宏的形式来声明和实现消息映射表。

在类的声明中用DECLARE\_MESSAGE\_MAP宏来声明消息映射表。

在类的声明外，使用BEGIN\_MESSAGE\_MAP和END\_MESSAGE\_MAP一对宏来定义消息映射表的节点，并且往节点中插入数据。我们经常会看到在这两个宏之间有一些ON\_XXX宏的定义，这些宏就是向消息映射表中插入表项。END\_MESSAGE\_MAP宏表示消息映射表的结束。

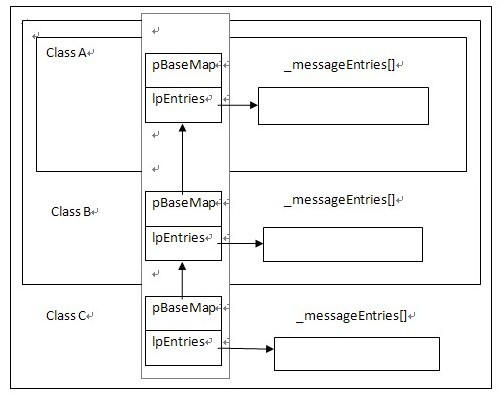


图2-4类族的消息映射表

在基于MFC的应用程序中，所有需要响应消息的类都派生于CCmdTarget类，所有这些类就形成了一个类族。所有类的消息映射表会在CCmdTarget类中形成一个总的消息映射表。为了使类族的所有子类能够将自己的消息映射表添加到总的消息映射表，MFC要求这个CCmdTarget类族的每一个类都包含一个链表的节点，这个节点由lpEntriesp和pBaseMap两个指针组成。前一个指针指向本类的消息映射表，后一个指针指向基类的节点，由此就形成了一个单项链表，该链表包含该类族所有类的消息映射表。如图2-4展示了一个类族的消息映射表。

由于这个总的消息映射表包含当前应用程序中的消息和消息响应函数之间的一个对应关系，因此这个总的消息映射表就是应用程序的消息映射表。有了这个消息映射表以后，CCmdTarget类族中任何一个类对象接收的消息，都可以在这个消息映射表中找到与之对应的消息处理函数。

## MFC文档/视图框架中各个对象的关系

在基于MFC的应用程序开发中，绝大多数Windows应用程序都是使用的文档/视图框架结构。这种框架结构就是将传统的应用程序主窗口对象拆分成三个部分，这三部分分别是框架窗口类CFrameWnd对象、视图类CView对象和文档类CDocument对象。这种拆分方式不仅大大减轻了窗口对象的负担和使得各个模块的分工更加的明确以及各个对象的任务更加的专一，而且大大方便了代码的编写和维护工作。其中，窗口框架类CFrameWnd负责管理窗口框的大小、标题和菜单栏以及状态栏等部分；文档类CDocument负责数据的存储和管理；视图类Cview负责数据的显示和用户的交互。

为了使程序框架的各个对象之间能够取得联系，文档、视图和窗口框架对象都从其基类继承了获取其他对象指针的方法。这样就可以在一个对象里的到其他对象的指针。比如，视图对象可以使用GetDocument()方法获得与其关联的文档对象指针；使用GetParentFrame()获得所属的框架窗口对象。

如果想要在框架窗口对象里面获取活动视图对象的指针可以使用GetActiveView()函数；要想获得视图对象的文档可以使用GetActiveDocument()函数。

如果想要在文档对象里获取视图链表的视图对象，就需要使用GetFirstViewPosition()函数和GetNextView()函数。图2-5展示了各个对象之间的联系。

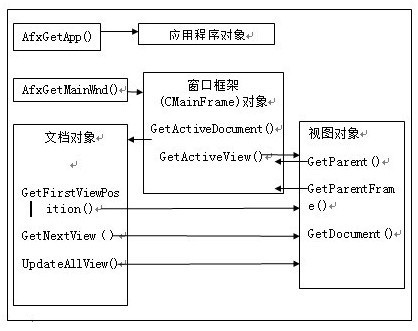


图2-5应用程序框架各对象之间的联系方法

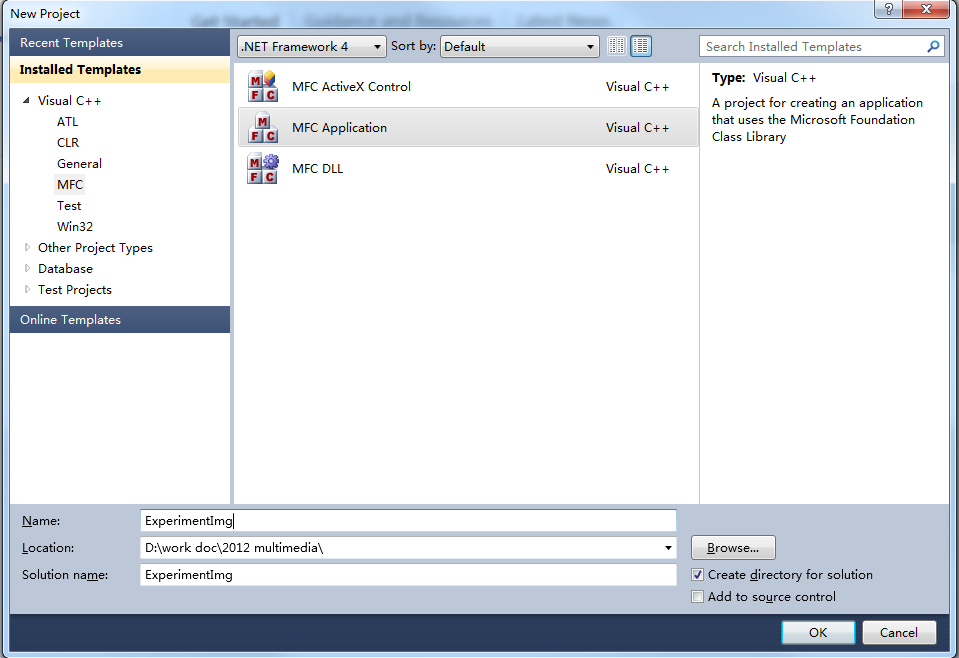
## 建立窗口程序示例

**实验任务2 可视化程序开发：**按照下述过程完整创建一个可视化程序

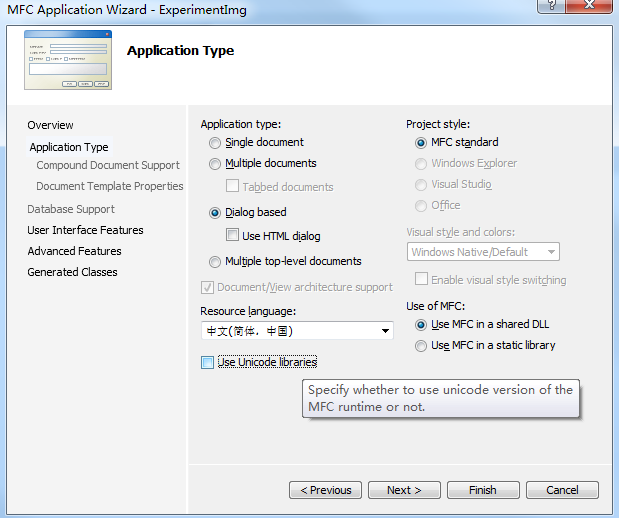
关键机制：

控件->添加成员变量->添加事件函数

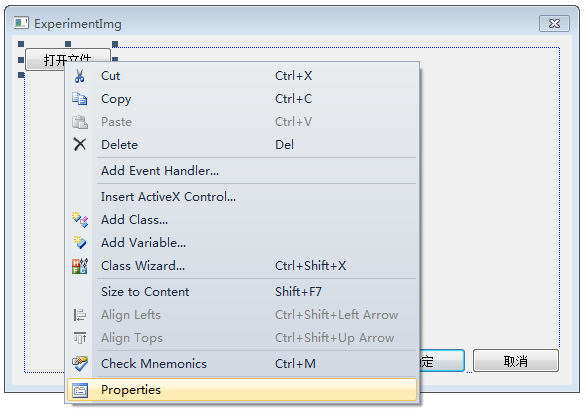
获取窗体句柄->根据控件ID获取控件句柄->获取成员变量



创建MFC应用程序

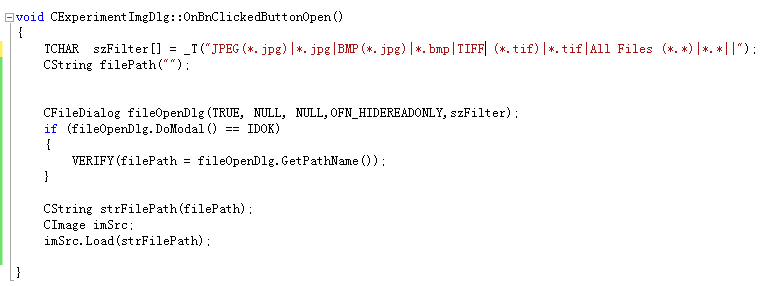


基于对话框的程序类型（非unicode）



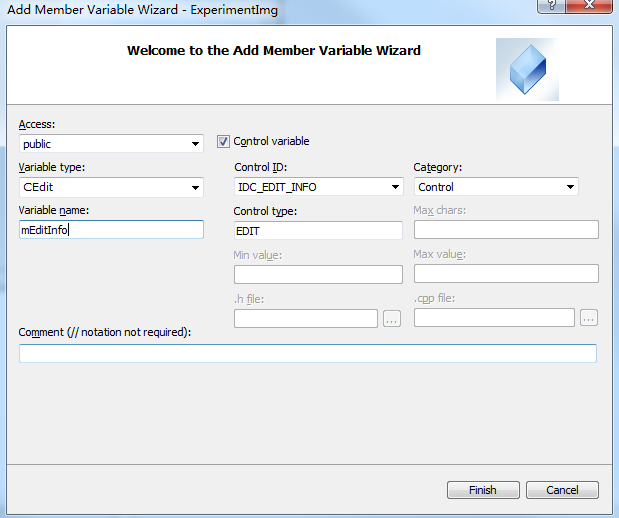
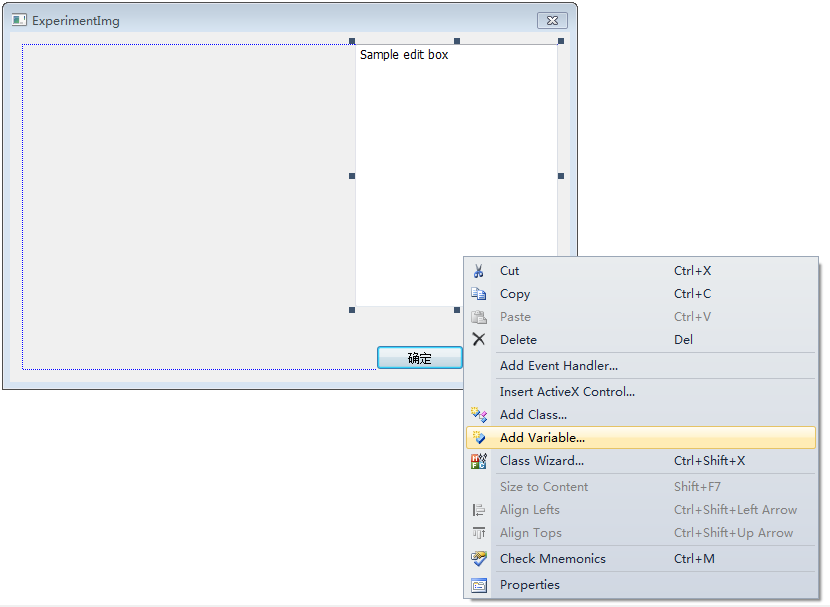
创建一个按钮，输入“打开文件”，并且重命名ID为IDC\_BUTTON\_OPEN

双击按钮，添加打开文件的函数：



注意，这里使用了CFileDialog类。如果是要选择目录，则使用API函数SHBrowseForFolder以及BROWSEINFO结构体的指针做参数变量。

接下来，创建一个EditBox，重命名为IDC\_EDIT\_INFO，并且添加成员变量



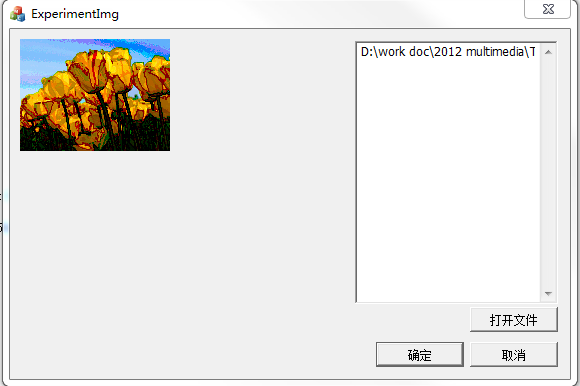
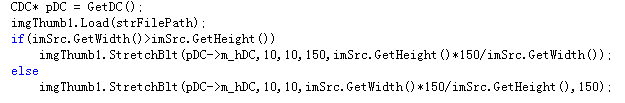
在源代码中查找”medit”可以看出，成员变量与控件被关联起来：



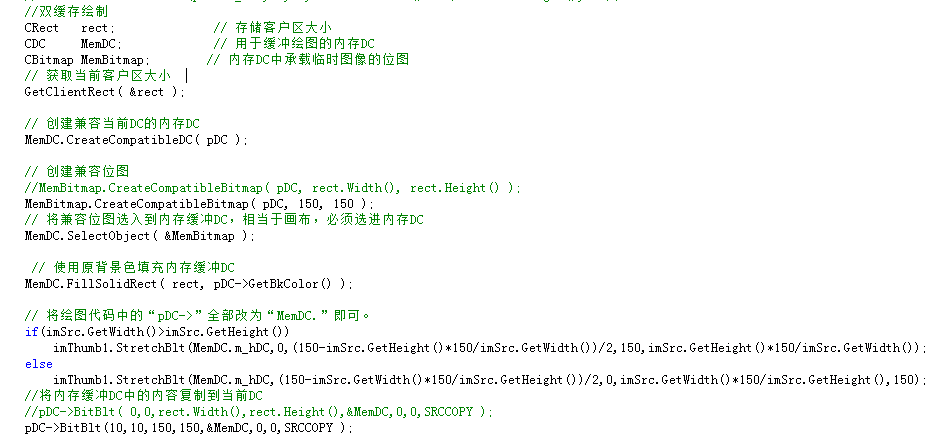
在对话框中，追加文件打开信息：

mEditInfo.SetWindowTextA(strFilePath + "\n");

最后，在对话框上绘制该图像的缩略图：



这是直接绘制。实际上对PDC的操作，更多使用双缓存技术：



# 三 多线程图像处理程序设计

**实验任务3 多线程图像处理基础：**

已有程序例子，采用MFC多线程开发方式实现如下功能

对一张大图像，使用8个线程分区块同时进行自适应中值滤波，采用合适的界面展示滤波结果，并输出单线程与多线程处理的时间。

使用8个线程对图像添加椒盐噪声，添加完成以后，再对其分8个线程进行自适应中值滤波。

阅读程序框架，继续采用Windows多线程和OpenMP两种方式，补充实现下述功能：

采用三阶插值的图像任意角度旋转与缩放（参考材料3.2.1）

自动色阶与自动白平衡（参考材料3.3.2）

图像素描转换（参考材料3.4.4）

参考材料链接:http://pan.baidu.com/s/1mgMaaxU 密码:2rpa

**实验任务4 图像融合与双边滤波：**

1. 采用opencv与boost库多线程开发方式，实现一个图像融合的程序，功能要求描述如下：

图像1与图像2等大小，像素分别为P1，P2。采用alpha参数，每个点的像素值，等于alpah\*P1 + （1-alpha）\*P2，界面上显示滑动条拖动alpha从0到1时，图像融合的效果。

2. 基于opencv的自适应双边滤波器函数，实现对图像的保边缘滤波。参考该算法，改造成基于boost库多线程的实现方式。

# 四 实验结果提交

任务3、4的源代码、可执行程序，以及word文档实验报告：

实验报告描述实验任务3和4的实现算法，对应部分的代码解释。

提交时间：2016-10-12