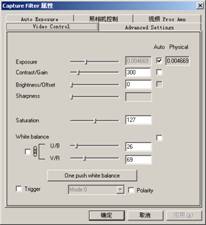
基于 DirectShow 视频及图片捕获软件的开发

我们知道目前很多工业相机的图像数据采集都是基于 DirectShow 的，常见的有映美精等。 DirectShow 是微软公司提供的一套在 Windows 平台上进行流媒体处理的开发包，与 DirectX 开发包一起发布。 DirectShow 为 多媒体 流的捕捉和回放提供了强有力的支持。运用 DirectShow ，我们可以很方便地从支持 WDM 驱动模型的采集卡上捕获数据，并且进行相应的后期处理乃至存储到文件中。它广泛地支持各种媒体格式，包括 Asf 、 Mpeg 、 Avi 、 Dv 、 Mp3 、 Wave 等等，使得多媒体数据的回放变得轻而易举。另外， DirectShow 还集成了 DirectX 其它部分（比如 DirectDraw 、 DirectSound ）的技术，直接支持 DV D 的播放，视频的 非线 性编辑，以及与数字摄像机的数据交换。更值得一提的是， DirectShow 提供的是一种开放式的开发环境，我们可以根据自己的需要定制自己的组件。

笔者使用 visual studio 2005 来开发了基于 DirectShow 的视频捕获软件，并用开发的软件对映美精相机进行了测试 。本软件不但可以实现对相机的视频捕获，而且还可以抓取图像帧。软件运行时自动搜索所连接的相机，预览后可以对相机参数进行设置。下面是软件的主界面。



   预览视频后可以对视频格式和图像参数进行设置。开始预览时，捕获的视频是黑白的，我们将颜色空间设置为 UYVY 即可捕获彩色视频。



下面是捕获的一帧图像，图像质量虽然没有映美精自带的软件效果好，但已经实现了所需各项基本功能，接下来的工作将会进一步提高软件性能。



另外我们还可以捕获视频，点击“捕获视频”按钮，输入要保持的文件名，注意要以 .avi 后缀结尾，点确定就开始捕获视频。

从我们开发的软件可以看到，映美精的相机能够很好的支持 DirectShow 的驱动，我们的软件对映美精相机的识别是如此的容易。接下来我们将继续开发基于其它驱动的图像捕获软件，为最终实现在一个软件中识别各种相机而努力。我们将逐步开放我们的源代码，以便更多的同行一起来探讨相机的图像采集技术。

下面是详细的软件开发过程。

一、 安装 DirectShow 和 visual studio 2005

首先我们安装 DirectShow SDK ，它有许多版本，作者使用的是 2003 年发布的 dx90bsdk.exe ，安装在 D 盘的 DXSDK 下。软件下载地址为 http://download.microsoft.com/download/b/6/a/b6ab32f3-39e8-4096-9445-d38e6675de85/dx90bsdk.exe

然后安装好 visual studio 2005 。安装完以后我们将进行开发环境的配置。

二、开发环境配置

开发环境的配置主要有两个工作要做：一是在使用 Directshow SDK 开发自己的程序时需要的 DirectShow 的有关静态库的配置，二是 visual C++ 开发环境的配置。

1、 生成 DirectShow SDK 开发库

使用 DirectShow SDK 开发用户自己的程序需要几个静态链接库： quartz.lib 、 strmbasd.lib 、 STRMBASE.lib 和 strmiids.lib 。中间两个 lib 需要用户自己编译生成，而其他两个微软已经提供。下表列出了使用 DirectShow SDK 开发程序所有要使用的库。

|  |  |
| --- | --- |
| 库名 | 功能说明 |
| Strmiids.lib | 定义了 DirectShow 标准的输出类标识（ CLSID ）和接口标识（ IID ） |
| Strmbasd.lib | 流媒体开发用到的库， Debug 、 Debug\_Unicode 版本 |
| Strmbase.lib | 流媒体开发用到的库， Release 、 Release\_Unicode 版本 |
| Quartz.lib | 定义了导出函数 AMGetErrorText |
| Winmm.lib | 使用 Windows 多媒体编程用到的库 |
|  |  |

基于 VC++2005 开发软件使用 DirectShow SDK ，首先需要用户编译 DirectShow 自带的源代码工程 baseclasses, 以生成 DirectShow SDK 的不同版本的库。同时由于 DirectShow SDK 是早期的 VC 开发软件，所以使用 VC++2005 编译 DirectShow SDK 会出现很多编译问题。下面列出了详细的编译过程和问题分析、解决方法。

1.1 编译工程 baseclasses 工程

启动 VS2005 ，选择“文件”→“打开”→“项目 / 解决方案”命令，在弹出的对话框中打开“ BaseClasses ”项目。

打开“ baseclasses.sln ”项目。如果 VS2005 有提问，则默认同意或确定。现在就开始编译该项目。按“ F7 ”快捷键可以编译生成项目。初次编译 VS2005 会报很多错误或者警告，有的需要我们手工修改程序，或者修改 VS2005 环境配置或编译选项；有的是一类问题，解决方法也有很多种。具体解决方法请参考路锦正的 《 Visual C++ 音频 / 视频处理技术及工程实践》第 225 页 -229 页。

1.2 Visual C++ 开发环境配置

有了 DirectShow SDK 库，用户就可以使用这些库来开发自己的程序了。为了能让 VC++ 自动搜寻到 SDK 库和头文件，还需要对 VC++ 的开发环境进行配置。添加库或路径的时候，根据你的要求添加 Debug 、 Release 、 Debug\_Unicode 、 Release\_Unicode 版本的库所在路径。下面假定添加非 Unicode 版本的库或路径。

首先确定 VC2005 是否已经包含了库和头文件所在的路径，因为在安装 VC2005 时，它会自动添加该目录。如果没有，则需要用户手工添加。

1.       更改添加的 include 内容：

D:\DXSDK\Include

D:\DXSDK\Samples\C++\DirectShow\BaseClasses

D:\DXSDK\Samples\C++\Common\Include

添加过程如下。选择“工具”→“选项”命令，在“项目和解决方案下”选择“ VC++ 目录”，在下拉框中选择“包含文件”选项，将上面的三个 Include 内容添加进去。

2.       更改添加 lib 路径

要添加的 lib 内容：

D:\DXSDK\Lib

D:\DXSDK\ Samples\C++\DirectShow\BaseClasses\Debug

D:\DXSDK\ Samples\C++\DirectShow\BaseClasses\Debug\_Unicode

D:\DXSDK\ Samples\C++\DirectShow\BaseClasses\Release

D:\DXSDK\ Samples\C++\DirectShow\BaseClasses\Release\_Unicode

添加过程和 Include 内容相似，只是在下拉框中选择“库文件”选项。

3.       添加链接库支持

上面的设置是在 VC2005 的开发环境的目录（ Directories ）中，添加用户在开发中可能用到的库或头文件“路径”，需要明确的事文件夹，而不是具体的文件。所以，要使用相关的库支持，还要用户明确地把要使用的库包含、添加到开发环境中。

基于 DirectShow SDK 开发流媒体应用程序，一般需要链接 strmiids.lib 和 quartz.lib, 前者定义了 DirectShow 标准的类标识符 CLSID 和接口标识 IID ，后者定义了导出函数 AMGetErrorText( 如果应用程序中没有使用这个函数，也可以不链接这个库 ) 。

在编译生成 DirectShow 的 BaseClasses 库 strmbasd.lib 、 STRMBASE.lib 时，由于该工程是生成库而不是应用程序，所以在编译该项目时 VC++2005 没有“链接器”选项。但是在开发其他应用可执行程序时，需要添加 DirectShow SDK 库的支持。添加路径：项目→属性→配置属性→链接器→输入→附加依赖项，输入 strmiids.lib quartz.lib ，库名之间用空格分开。另外，在程序中使用 DirectShow SDK 类或接口的代码程序中，还要添加 #include<dshow.h> 。

在添加链接库时，除了以上配置 VC 的开发环境外，也可以在源程序文件开头部分，直接语句编程引入 #pragma comment(lib, ” strmiids.lib ” ) 。

如果程序中没有使用 dshow.h ，而是包含了 stream.h, 则库文件需要链接 strmbased.lib 、 winmm.lib ，在源程序文件开头添加：

#pragma comment(lib, ” strmbasd.lib ” )

#pragma comment(lib, ” winmm.lib ” )

#include <streams.h>

不过，编译器会报出以下的错误。

error C2146: 语法错误为缺少“；”（在标识符“ m\_pString ”的前面）。

问题定位在 wxdebug.h(329) 中。经分析得知，由于某种原因，编译器认为 PTCHAR 没有定义，那用户可以在类外定义： typedef WCHAR \*PTCHAR; 再编译项目。

三、开发过程

DirectShow SDK 的视频采集经典技术是使用 ICaptureGraphBuilder2 标准接口，利用其方法 RenderStream 自动建立、连接滤波器链表。 RenderStream 方法在预览、捕获视频时引脚的类型分为 PIN\_CATEGORY\_PREVIEW 和 PIN\_CATEGORY\_CAPTURE ，媒体类型均为 MEDIATYPE\_Video 。此实例要完成的目的有两个：一是实时预览采集的视频数据；二是在预览图像的同时，实时地把捕获数据保存到文件中。首先我们使用 GraphEdit 模拟实现该过程。

1、 GraphEdit 模拟实现

步骤一、 添加 "Video Capture Sources" 视频捕获设备，如图 1 所示。

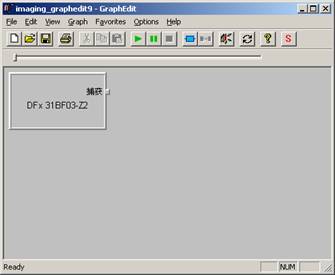


图 1 、添加视频捕获设备

步骤二、视频捕获滤波器只有一个 Pin ，而我们要求在预览数据的同时还能够保存数据，即需要一个组件把捕获的流分成两个。 DirectShow SDK 为此提供了 Smart Tee 滤波器，把捕捉的视频流分成两个流供使用。在 GraphEdit 中单击 "DirectShow Filters" 按钮，插入 "Smart Tee" 滤波器，如图 2 所示

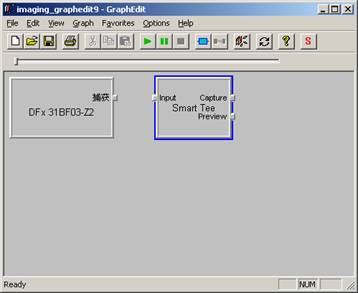


图 2 添加 Smart Tee 滤波器

步骤三、采集捕捉的视频数据保存到文件，以 AVI 格式写文件。插入 "AVI Mux" 滤波器，如图 3 所示。

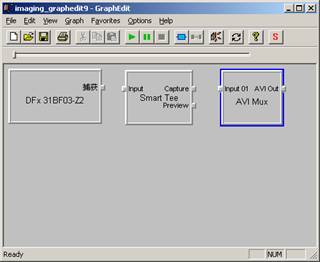


图 3 添加 AVI Mux 滤波器

步骤四、插入 "File writer" 滤波器，保存文件命名为 "a.avi" 。如图 4 所示

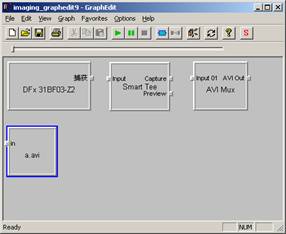


图 4 插入 File writer

步骤五、插入 "SampleGrabber" 和 "Video Renderer" 滤波器，如图 5 所示

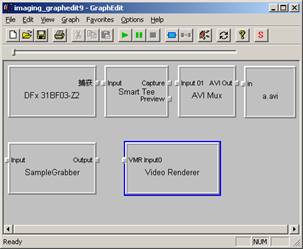


  图 5 插入 SampleGrabber 和 Video Renderer 滤波器

步骤六、 最后把所有的滤波器用鼠标连接起来，完成构建滤波器链表，如图 6 所示

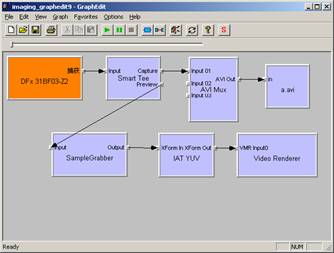


图 6 视频预览、保存滤波器链表

步骤七、运行滤波器链表，单击 "Graph"→"Play" 按钮执行视频数据的预览、保存。

1、 视频捕获类 CCaptureClass 的实现

详细讲述 CCaptureClass 类的成员变量和其他成员方法的实现，剖析其完成视频采集、保存的技术过程。

1 ）定义 CCaptureClass 类

|  |
| --- |
| class CCaptureClass { public: CCaptureClass();          // 类构造器 virtual ~CCaptureClass(); // 类析构器 |
|  |

|  |
| --- |
| int EnumDevices(HWND hList); void SaveGraph(TCHAR \*wFileName);            // 保存滤波器链表 void ConfigCameraPin(HWND hwndParent);       // 配置摄像头的视频格式  void ConfigCameraFilter(HWND hwndParent);     // 配置摄像头的图像参数  BOOL Pause(); // 暂停  BOOL Play(); // 播放  BOOL Stop(); // 停止  HRESULT CaptureImages(CString inFileName);    // 捕获保存视频  BOOL CaptureBitmap( const char \* outFile); // 捕获图片 |
| HRESULT PreviewImages(int iDeviceID, HWND hWnd); // 采集预览视频 private: HWND       m\_hWnd;          // 视频显示窗口的句柄 IGraphBuilder    \*m\_pGB;          // 滤波器链表管理器 ICaptureGraphBuilder2  \*m\_pCapture;  // 增强型捕获滤波器链表管理器 IBaseFilter     \*m\_pBF;   // 捕获滤波器  IBaseFilter      \*pNull ;   // 渲染滤波器  IBasicVideo \*pBasicVideo;// 视频基本接口  IBaseFilter \*pGrabberF;// 采样滤波器  ISampleGrabber \*pGrabber;// 采样滤波器接口 IMediaControl    \*m\_pMC;   // 媒体控制接口  IMediaEventEx \*pEvent; // 媒体事件接口 IVideoWindow     \*m\_pVW;   // 视频显示窗口接口 IBaseFilter     \*pMux;   // 写文件滤波器 protected: bool BindFilter(int deviceId, IBaseFilter \*\*pFilter);    // 把指定的设备滤波器捆绑到链表中 void ResizeVideoWindow();               // 更改视频显示窗口 HRESULT SetupVideoWindow();             // 设置视频显示窗口的特性 HRESULT InitCaptureGraphBuilder();     // 创建滤波器链表 管理器，查询其各种控制接口 |
|  |

上述代码是类 CCaptureClass 的成员变量和成员函数，成员变量包括 DirectShow 开发流媒体播放应用程序需要的各种接口指针变量。成员函数实现创建滤波器链表管理器、配置视频采集格式、配置图像参数和保存滤波器链表等。

在类的构造器和析构器中完成对

|  |
| --- |
| \* 定义的资源释放操作宏 \*/ #ifndef srelease #define srelease(x) |
|  |

|  |
| --- |
| if ( NULL != x ) {  x->Release( );  x = NULL;  } #endif /\* 类构造函数实现 \*/ CCaptureClass::CCaptureClass() { CoInitialize(NULL);  //COM 库初始化 m\_hWnd = NULL;    // 视频显示窗口的句柄 m\_pVW = NULL;    // 视频窗口接口指针清空 m\_pMC = NULL;    // 媒体控制接口指针清空 m\_pGB = NULL;   // 滤波器链表管理器接口指针清空 m\_pBF = NULL;     // 捕获滤波器接口指针清空  pBasicVideo = NULL;// 基类视频接口指针清空  pGrabberF = NULL; // 采样滤波器接口指针清空  pNull = NULL; // 渲染滤波器接口清空  pGrabber = NULL; //  pEvent = NULL; // 媒体事件接口指针清空 m\_pCapture = NULL;   // 增强型捕获滤波器链表管理器接口指针清空 } /\* 析构函数实现 \*/ CCaptureClass::~CCaptureClass() { if (m\_pMC)  m\_pMC->Stop();   // 首先停止媒体 if (m\_pVW) { m\_pVW->put\_Visible(OAFALSE); // 视频窗口不可见 m\_pVW->put\_Owner(NULL);  // 视频窗口的父窗口清空 } srelease(m\_pCapture);       // 释放增强型捕获滤波器链表管理器接口 srelease(pBasicVideo);  srelease(pGrabber);  srelease(pGrabberF);  srelease(pNull);  srelease(pEvent); srelease(m\_pMC);       // 释放媒体控制接口 srelease(m\_pGB);       // 释放滤波器链表管理器接口 srelease(m\_pBF);      // 释放捕获滤波器接口 CoUninitialize();     // 卸载 COM 库 |
|  |

在类构造函数中，清空所有指针以便于清楚其后续操作的状态。析构函数释放各种资源、指针并清空指针，最后卸载 COM 库。

2 ）根据指定的设备 ID ，把基本滤波器与设备捆绑

首先枚举系统所有采集设备，直到列举的 ID 相同为止，最后 BindToObject 完成捆绑。

|  |
| --- |
| // 把指定采集设备与滤波器捆绑 bool CCaptureClass::BindFilter(int deviceId, IBaseFilter \*\*pFilter) { if (deviceId < 0) return false; // 枚举所有的视频捕获设备 ICreateDevEnum \*pCreateDevEnum; // 生成设备枚举器 pCreateDevEnum HRESULT hr = CoCreateInstance(CLSID\_SystemDeviceEnum, NULL,  CLSCTX\_INPROC\_SERVER, IID\_ICreateDevEnum, (void\*\*)&pCreateDevEnum); if (hr != NOERROR)  return false; IEnumMoniker \*pEm; // 创建视频输入设备类枚举器 hr = pCreateDevEnum->CreateClassEnumerator (CLSID\_VideoInputDeviceCategory,  &pEm, 0); if (hr != NOERROR) return false; pEm->Reset();         // 复位该设备 ULONG cFetched; IMoniker \*pM; int index = 0; // 获取设备 while(hr = pEm->Next(1, &pM, &cFetched), hr==S\_OK, index <= deviceId) { IPropertyBag \*pBag; // 获取该设备的属性集 hr = pM->BindToStorage(0, 0, IID\_IPropertyBag, (void \*\*)&pBag); if(SUCCEEDED(hr))  { VARIANT var; var.vt = VT\_BSTR;     // 保存的是二进制的数据 hr = pBag->Read(L"FriendlyName", &var, NULL);          // 获取 FriendlyName 形式的信息 if (hr == NOERROR)  { // 采集设备与捕获滤波器捆绑 if (index == deviceId) pM->BindToObject(0, 0,  IID\_IBaseFilter,         (void\*\*)pFilter); SysFreeString(var.bstrVal); // 释放二进制数据资源，必须释放 |
|  |

|  |
| --- |
| }    pBag->Release(); } pM->Release(); index++; } return true; } |
|  |

该函数的传入参数是采集设备的索引号和捕获设备的滤波器。根据索引号查询系统中的视频捕获设备。以友好名字（ FriendlyName ）的方式获取所选设备的信息，然后把查询成功的设备与传入的滤波器捆绑，返回捕获设备的滤波器。

3 ）设置视频显示窗口

DirectShow 的显示窗口与 IVideoWindow 接口的设置基本相同，把传入的显示窗口的句柄捆绑到 IvideoWindow 接口上。

|  |
| --- |
| /\* 设置视频显示窗口的特性 \*/ HRESULT CCaptureClass::SetupVideoWindow() { HRESULT hr; //m\_hWnd 为类 CCaptureClass 的成员变量，在使用该函数前须初始化 hr = m\_pVW->put\_Visible(OAFALSE);                  // 视频窗口不可见 hr = m\_pVW->put\_Owner((OAHWND)m\_hWnd);      // 窗口所有者：传入的窗口句柄 if (FAILED(hr)) return hr; hr = m\_pVW->put\_WindowStyle(WS\_CHILD | WS\_CLIPCHILDREN);// 设置窗口类型 if (FAILED(hr)) return hr; ResizeVideoWindow();                           // 更改窗口大小 hr = m\_pVW->put\_Visible(OATRUE);              // 视频窗口可见 return hr; } /\* 更改视频窗口大小 \*/ void CCaptureClass::ResizeVideoWindow() { if (m\_pVW) { // 让图像充满整个指定窗口 CRect rc; ::GetClientRect(m\_hWnd,&rc);    // 获取显示窗口的客户区 m\_pVW->SetWindowPosition(0, 0, rc.right, rc.bottom); // 设置视频显示窗口的位置 }  } |
|  |

在调用该函数前，需要把应用程序的显示窗口句柄传入以初始化 m\_hWnd 。首先视频窗口不可见，捆绑传入的窗口句柄为视频窗口，设置窗口类型。 ResizeVideoWindow 函数获取显示窗口的客户区域，利用视频窗口接口的方法 SetWindowPosition 设置视频显示窗口的位置。

4 ）预览采集到的视频数据

使用上述有关的类成员函数初始化滤波器链表管理器，把指定采集设备的滤波器添加到链表中，然后渲染 RenderStream 方法把所有的滤波器链接起来，最后根据设定的显示窗口预览采集到的视频数据，具体实现过程如下。

|  |
| --- |
| /\* 开始预览视频数据 \*/ HRESULT CCaptureClass::PreviewImages(int iDeviceID, HWND hWnd) {  HRESULT hr;    // 初始化视频捕获滤波器链表管理器  hr = InitCaptureGraphBuilder();  if (FAILED(hr))  {  AfxMessageBox(\_T( "Failed to get video interfaces!" ));  return hr;  }  // 把指定采集设备与滤波器捆绑  if (!BindFilter(iDeviceID, &m\_pBF))  return S\_FALSE;  // 把滤波器添加到滤波器链表中  hr = m\_pGB->AddFilter(m\_pBF, L "Capture Filter" );  if ( FAILED( hr ) )  {  AfxMessageBox(\_T( "Can ’ t add the capture filter" ));  return hr;  }  // Create the Sample Grabber.  hr = CoCreateInstance(CLSID\_SampleGrabber,NULL, CLSCTX\_INPROC\_SERVER,  IID\_IBaseFilter, ( void \*\*)&pGrabberF);  if ( FAILED( hr ) )  {  AfxMessageBox(\_T( "Can ’ t create the grabber" ));  return hr;  }  hr = pGrabberF->QueryInterface(IID\_ISampleGrabber, ( void \*\*)&pGrabber);  // 把滤波器添加到滤波器链表中 |
|  |

|  |
| --- |
| hr = m\_pGB->AddFilter(pGrabberF, L "Sample Grabber" );  if ( FAILED( hr ) )  {  AfxMessageBox(\_T( "Can ’ t add the grabber" ));  return hr;  }  // Add the Null Renderer filter to the graph.  hr = CoCreateInstance(CLSID\_VideoRenderer, NULL, CLSCTX\_INPROC\_SERVER,  IID\_IBaseFilter, ( void \*\*)&pNull);  hr = m\_pGB->AddFilter(pNull, L "VideoRender" );  if ( FAILED( hr ) )  {  AfxMessageBox(\_T( "Can ’ t add the VideoRender" ));  return hr;  }  // 渲染媒体，把链表中滤波器连接起来  hr = m\_pCapture->RenderStream( &PIN\_CATEGORY\_PREVIEW, &MEDIATYPE\_Video, m\_pBF, pGrabberF, pNull);  if ( FAILED( hr ) )  {  AfxMessageBox(\_T( "Can ’ t build the graph" ));  return hr;  } // 设置视频显示窗口 m\_hWnd = hWnd;         // 初始化窗口句柄 SetupVideoWindow();   // 设置显示窗口 hr = m\_pMC->Run();   // 开始采集、预览视频，在指定窗口显示视频 if(FAILED(hr)) { AfxMessageBox(\_T("Couldn't run the graph!")); return hr; } return S\_OK; } |
|  |

上述程序从最初的创建滤波器链表管理器、枚举系统视频采集设备、把采集设备与捕获滤波器捆绑，到添加滤波器到滤波器链表、设置视频显示窗口，最后开始运行媒体：采集、预览视频，包含了使用 DirectShow SDK 开发视频采集、预览的整个技术过程。函数功能独立而又完整。

5 ）保存采集到的数据

把捕获的视频以 AVI 格式写文件。注意设置前停止调用滤波器链表，设置完成后再运行链表。

|  |
| --- |
| /\* 设置捕获视频的文件，开始捕捉视频数据写文件 \*/ HRESULT CCaptureClass::CaptureImages(CString inFileName) { |
|  |

|  |
| --- |
| HRESULT hr=0; m\_pMC->Stop();      // 先停止视频 // 设置文件名，注意第二个参数的类型 hr = m\_pCapture->SetOutputFileName( &MEDIASUBTYPE\_Avi,  inFileName.AllocSysString(), &pMux, NULL ); // 渲染媒体，链接所有滤波器 hr = m\_pCapture->RenderStream( &PIN\_CATEGORY\_CAPTURE,   &MEDIATYPE\_Video, m\_pBF, NULL, pMux ); pMux->Release(); m\_pMC->Run();     // 回复视频 return hr; } |
|  |

预览视频后，用户可以使用该函数存储捕获的视频数据。首先停止视频媒体，利用 ICaptureGraphBuilder2 的方法 SetOutputFileName 设置存储捕获数据的文件名，然后渲染视频媒体， RenderStream 方法自动链接图表中的滤波器，最后开始运行视频媒体。

6 ）捕获图片

把捕获的视频以 bmp 格式写文件。我们使用使用 Sample Grabber filter 抓取图像。 sample Grabber 使用两种模式抓取图像：缓冲模式和回调模式，缓冲模式向下传递采样时拷贝每个采样，而回调模式对于每个采样调用程序定义的回调函数。我们采用缓冲模式。

|  |
| --- |
| BOOL CCaptureClass::CaptureBitmap( const char \* outFile) //const char \* outFile)  {  HRESULT hr=0;  // 取得当前所连接媒体的类型  AM\_MEDIA\_TYPE mt;  hr = pGrabber->GetConnectedMediaType(&mt);  // Examine the format block.  VIDEOINFOHEADER \*pVih;  if ((mt.formattype == FORMAT\_VideoInfo) &&  (mt.cbFormat >= sizeof (VIDEOINFOHEADER)) &&  (mt.pbFormat != NULL) )  {  pVih = (VIDEOINFOHEADER\*)mt.pbFormat;  }  else  {  // Wrong format. Free the format block and return an error.  return VFW\_E\_INVALIDMEDIATYPE;  }  // Set one-shot mode and buffering.  hr = pGrabber->SetOneShot(TRUE); |
|  |

|  |
| --- |
| if (SUCCEEDED(pGrabber->SetBufferSamples(TRUE)))  {  bool pass = false ;  m\_pMC->Run();  long EvCode=0;  hr = pEvent->WaitForCompletion( INFINITE, &EvCode );  //find the required buffer size  long cbBuffer = 0;  if (SUCCEEDED(pGrabber->GetCurrentBuffer(&cbBuffer, NULL)))  {  //Allocate the array and call the method a second time to copy the buffer:  char \*pBuffer = new char [cbBuffer];  if (!pBuffer)  {  // Out of memory. Return an error code.  AfxMessageBox(\_T( "Out of Memory" ));  }  hr = pGrabber->GetCurrentBuffer(&cbBuffer, ( long \*)(pBuffer));  // 写到 BMP 文件中  HANDLE hf = CreateFile(LPCTSTR(outFile), GENERIC\_WRITE, FILE\_SHARE\_WRITE, NULL, CREATE\_ALWAYS, 0, NULL);  if (hf == INVALID\_HANDLE\_VALUE)  {  return 0;  }  // Write the file header.  BITMAPFILEHEADER bfh;  ZeroMemory(&bfh, sizeof (bfh));  bfh.bfType = 'MB' ; // Little-endian for "MB".  bfh.bfSize = sizeof ( bfh ) + cbBuffer + sizeof (BITMAPINFOHEADER);  bfh.bfOffBits = sizeof ( BITMAPFILEHEADER ) + sizeof (BITMAPINFOHEADER);  DWORD dwWritten = 0;  WriteFile( hf, &bfh, sizeof ( bfh ), &dwWritten, NULL );  // Write the bitmap format  BITMAPINFOHEADER bih;  ZeroMemory(&bih, sizeof (bih));  bih.biSize = sizeof ( bih ); |
|  |

|  |
| --- |
| bih.biWidth = pVih->bmiHeader.biWidth;  bih.biHeight = pVih->bmiHeader.biHeight;  bih.biPlanes = pVih->bmiHeader.biPlanes;  bih.biBitCount = pVih->bmiHeader.biBitCount;  dwWritten = 0;  WriteFile(hf, &bih, sizeof (bih), &dwWritten, NULL);  //write the bitmap bits  dwWritten = 0;  WriteFile( hf, pBuffer, cbBuffer, &dwWritten, NULL );  CloseHandle( hf );  pass = true ;  }  return pass;  }  hr = pGrabber->SetOneShot(FALSE);  } |
|  |

预览视频后，用户可以使用该函数捕获图像。这个函数自动为捕获的图片命名并保存。

3 、 界面设计

本案例使用 VC++ 2005 的对话框应用程序框架设计视频捕获应用程序。

步骤一、应用 VC++ 2005 应用程序向导建立对话框程序框架，项目名称为 CaptureVideo 。

步骤二、在项目 CaptureVideo 的主界面中添加控件： 9 个 Button 、 1 个 Combo Box 、 2 个 Picture Control ，如表 1 所示。根据其功能修改所有控件的 ID 。

表 1  控件功能列表

|  |  |
| --- | --- |
| 名称 | 说明 |
| ID\_PREVIEW | 视频预览 |
| ID\_PAUSEPLAY | 暂停预览 |
| ID\_PLAY | 继续预览 |
| ID\_GRABIMAGE | 捕获图片 |
| ID\_CAPTURE | 捕获视频 |
| ID\_VIDEO\_FORMAT | 视频格式 |
| ID\_IMAGE\_PARAMETER | 图像参数 |
| ID\_SAVEGRAPH | 保存图表 |
| ID\_EXIT | 退出程序（终止预览捕获） |
| IDC\_DEVICE\_LISTER | 设备列表组合框 |
| IDC\_VIDEO\_WINDOW | 显示捕获的视频图像 |
| IDC\_LOGO | 显示公司 LOGO |
|  |  |

图 7 软件界面 计 图 8  控件右键菜单内容

添加控件变量后的 CCaptureVideoDlg 类的代码如下。

|  |
| --- |
| // 显示捕获的图像 CStatic m\_videoWindow; // 组合框列表，显示设备名称 CComboBox m\_listCtrl; |
|  |

步骤五、添加按钮在线提示 ToolTip 。

首先在类 CCaptureVideoDlg 定义中声明 tooltip 控件。

|  |
| --- |
| CToolTipCtrl m\_tooltip; |
|  |

接着在类 CCaptureVideoDlg 实现文件的对话框初始化函数 OnInitDialog 中添加：

|  |
| --- |
| m\_tooltip.Create(this); m\_tooltip.Activate(TRUE); m\_tooltip.AddTool(GetDlgItem(ID\_PREVIEW), \_T(" 开始预览视频 ")); // 添加其他按钮的 tooltip |
|  |

在 PreTranslateMessage 消息处理函数中添加如下代码，如果程序中没有该消息处理函数，则需要用户自己添加。

|  |
| --- |
| m\_tooltip.RelayEvent(pMsg); |
|  |

步骤六、功能按钮的消息响应，即单击按钮的事件处理。

双击某按钮，实现单击按钮事件处理函数的添加。为了使用类 CCaptureClass 的变量和函数，在 CCaptureVideoDlg 类中引入头文件并定义视频捕获类的对象。

首先引入头文件：

|  |
| --- |
| #include "CaptureClass.h" |
|  |

接着在 CCaptureVideoDlg 类中定义视频捕获类的对象：

|  |
| --- |
| CCaptureClass m\_cap; |
|  |

双击 " 视频预览 " 按钮，添加事件处理代码：

|  |
| --- |
| void CCaptureVideoDlg::OnBnClickedPreview() { //TODO: 在此添加控件通知处理程序代码 HWND hVWindow = m\_videoWindow.GetSafeHwnd();  // 获取视频显示窗口的句柄 int id = m\_listCtrl.GetCurSel();          // 获取当前选中的视频设备 m\_cap.PreviewImages(id , hVWindow);     // 开始预览视频 } |
|  |

获取视频显示窗口的句柄，根据程序最初枚举到的所有采集设备，选择用户选中的采集设备。调用视频捕获类的成员函数 PreviewImages 完成整个视频的采集、显示任务。

双击 " 视频捕获 " 按钮，添加事件处理代码。

|  |
| --- |
| void CCaptureVideoDlg::OnBnClickedCapture() { //TODO: 在此添加控件通知处理程序代码 CString strFilter = \_T("AVI File (\*.avi) | \*.avi|"); // 文件类型过滤器 strFilter += "All File (\*.\*) | \*.\*|"; CFileDialog dlg(TRUE, NULL, NULL,        // 打开另存为文件对话框 OFN\_PATHMUSTEXIST| OFN\_HIDEREADONLY, strFilter, this); if (dlg.DoModal() == IDOK) { CString m\_sourceFile = dlg.GetPathName(); // 获取用户输入的文件路径名 m\_cap.CaptureImages(m\_sourceFile);      // 开始捕获、存储视频 } } |
|  |

存储捕获的视频，这里采用 AVI 格式，视频数据未经压缩。调用视频捕获类 CCaptureClass 的成员函数 CaptureImages 完成视频的捕获、存储任务。本案例需要先预览视频，然后再开始捕获、存储视频，不能直接捕获、存储视频。

双击 " 视频格式 " 按钮，添加事件处理代码。

|  |
| --- |
| void CCaptureVideoDlg::OnBnClickedVideoFormat() { //TODO: 在此添加控件通知处理程序代码 m\_cap.ConfigCameraPin(this->m\_hWnd); } |
|  |

设置视频格式前，首先启动 " 预览视频 " ，然后再配置视频的格式。视频格式包括帧率、颜色空间、视频分辨率等，该功能的效果如图 9 所示。



图 9  视频格式配置属性页

双击 " 视频参数 " 按钮，添加事件处理代码。

|  |
| --- |
| void CCaptureVideoDlg::OnBnClickedImageParameter() { //TODO: 在此添加控件通知处理程序代码 m\_cap.ConfigCameraFilter(this->m\_hWnd);  } |
|  |

设置视频图像参数前，首先启动 " 预览视频 " ，然后配置图像参数。图像参数包括图像、白平衡、模式控制和去抖动等，该功能的效果如图 10 所示。并且该功能调整的参数马上起作用。

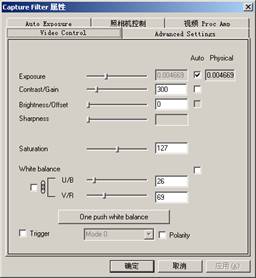


图 10  图像参数配置属性页

双击 " 捕获图片 " 按钮，添加事件处理代码。

|  |
| --- |
| void CCaptureVideoDlg::OnBnClickedGrabimage()  {  // TODO: 在此添加控件通知处理程序代码  static int c = 0;  TCHAR szFilename[MAX\_PATH];  DWORD dwPathLen = 0;  if ((dwPathLen = ::GetModuleFileName(::AfxGetInstanceHandle(),szFilename, MAX\_PATH ))== 0)  {  return ;  }  for ( int i=dwPathLen-1; i>=0; i--)  {  if (( '\\' == szFilename[i]) || ( '/' == szFilename[i]))  {  break ;  } else {  szFilename[i] = '\0' ;  }  }    CString str;  str.Format(\_T( "%s" ),szFilename);  CString strTemp;  strTemp.Format(\_T( "%d" ),c);  str += strTemp + \_T( ".bmp" ); |
|  |

|  |
| --- |
| c++;    TCHAR \*p=str.GetBuffer(str.GetLength());  str.ReleaseBuffer();  if (m\_cap.CaptureBitmap(( const char \*)p))  {  } else  MessageBox(\_T( " 抓图失败 !" ));  } |
|  |

本功能将自动给图片命名并设置保存路径。

双击 " 保存图表 " 按钮，添加事件处理代码。

|  |
| --- |
| void CCaptureVideoDlg::OnBnClickedSavegraph() { //TODO: 在此添加控件通知处理程序代码 CFileDialog dlg(TRUE); if (dlg.DoModal()==IDOK) { CString str=dlg.GetPathName();       // 要保存的 Graph 文件名 TCHAR \*inFileName = str.GetBuffer(str.GetLength());// 获取字符串指针 str.ReleaseBuffer();       // 切记要释放 Buffer  m\_cap.SaveGraph(inFileName);        // 保存 Graph } }  本功能存储的 Graph 文件可以使用程序 GraphEdit 播放。  双击 " 退出程序 " 按钮，添加事件处理代码。程序隐含调用了类 CCaptureClass 的析构函数，释放资源和 COM 库。 |
|  |

|  |
| --- |
| void CCaptureVideoDlg::OnBnClickedExit() { //TODO: 在此添加控件通知处理程序代码 CDialog::OnOK(); } |
|  |

退出本程序时，由于视频捕获类 CCaptureClass 的析构函数已经包含了释放资源、指针的工作，所以退出应用程序时不用释放任何资源，只是关闭应用程序。

其他工作

在对话框的初始化 OnInitDialog 中枚举本系统的视频采集设备，添加到列表框并默认显示第一个设备。

|  |
| --- |
| m\_cap.EnumDevices(m\_listCtrl.GetSafeHwnd()); m\_listCtrl.SetCurSel (0); |
|  |

至此我们详细介绍了软件的开发过程。在开发过程中我们遇到了许多问题，如对于图片捕获有很多种方法，不同的方法将在很大程度上影响软件的稳定性和捕获图片的效果。当然程序还存在许多有待改进的地方，我们将进一步完善它。关于本软件及其源代码，我们将于近期在中国视觉网上公布并提供下载，希望大家及时关注我们的网站 http://www.china-vision.net/ 。