# 视觉库说明

## 1. Visual Studio的编程设置

封装是编程的基本思想。能够完成特定功能的代码通常写成一组函数封装在一个DLL文件（动态库）或lib文件（静态库）中，组成一个功能模块，供应用程序调用。功能相关的多个模块可以组成模块集。例如Windows SDK是一组多语言的DLL，提供Windows编程所需的函数；OpenCV库是一组C++ DLL，实现计算机视觉功能。

C++的动态库除了DLL文件还有一个相应的lib文件。生成C++应用程序时外部功能模块由lib文件提供，动态库的lib文件仅包含DLL文件信息，生成的程序较小，需要DLL文件才能运行；静态库的lib文件包含模块所有代码，生成的程序较大，可以独立运行。在C#中没有静态库，DLL称为程序集，可以直接用于生成程序。C++ DLL和C# DLL不能随意混合调用。下面主要讨论动态库。

应用程序运行时必须能够找到所使用的DLL文件。DLL文件可以放在应用程序所在路径或系统路径，但建议放在系统路径。将DLL文件所在路径（例如D:\Utilities）加入系统路径步骤如下：

1）进入控制面板“系统属性”-“高级”-“环境变量”-“系统变量”-“path”。

2）在path变量最前端添加DLL文件所在路径，用分号“;”分隔（例如“D:\Utilities;”），注意不要删除原有内容。

3）重启计算机。

如果运行应用程序时提示缺少“xxx.dll”，通常是系统路径设置不正确，应检查DLL文件所在路径是否在系统路径中，按照上面的步骤再设置。

C++编程需要有三个文件：定义DLL函数的h或hpp文件，与DLL同时生成的lib文件，DLL文件。其中h/hpp文件和lib文件在生成程序时使用，DLL文件在运行程序时使用。用Visual Studio编写C++程序时，设置步骤如下：

1）进入项目属性，设置以下内容：

- C/C++。“常规”–“附加包含目录”：添加h文件所在的路径，例如D:\Utilities\includes。
- 链接器。“常规” – “附加库目录”：添加lib文件所在的路径，例如D:\Utilities。

2）在程序开头包含h文件和引用lib文件。以使用WinCVMat为例，代码如下：

```
#include "WinCVMat.h"
#pragma comment(lib,"WinCVMat.lib")
```

C# DLL包含了函数信息，编程仅需要DLL文件。用Visual Studio编写C#程序时，设置步骤如下：

1）进入“项目”-“添加引用”-“浏览”，找到所需DLL。

2）在程序开头用using说明DLL函数的命名空间。例如

```
using Vision;
```

# 2. OpenCV及其接口库说明

## 2.1. OpenCV 的配置

OpenCV 是开源视觉库，支持 C++, Python 和 Java 编程。要使用 OpenCV，首先进入官网 http://opencv.org，下载 OpenCV，直接解压。以解压到 D:\opencv 为例，库文件在 D:\opencv\build，源程序在 D:\opencv\sources。这里仅讨论 OpenCV 4.0 和 x64 程序，下面说明设置步骤。

### 2.1.1. 设置环境变量

打开系统属性-高级-环境变量，然后

1）创建系统变量 **OPENCV_DIR**，添加 OpenCV 路径 D:\opencv\build\x64\vc15。也可以用管理员权限打开 cmd，选择输入如下命令行：

| | |
|---|---|
| setx -m OPENCV_DIR D:\opencv\build\x64\vc14 | suggested for Visual Studio 2015 - 64 bit Windows |
| setx -m OPENCV_DIR D:\opencv\build\x64\vc15 | suggested for Visual Studio 2017 - 64 bit Windows |

2）在系统路径（系统变量 PATH）添加路径：“%OPENCV_DIR%\bin;”。注意用分号“;”分隔路径，不要删除原有内容。注意修改系统路径后需要重启计算机。

### 2.1.2. 设置 Visual Studio 项目属性

右键点击 C++项目，选择属性，然后选择：

1）C/C++。“常规”–“附加包含目录”：在 x64 项目添加“$(OPENCV_DIR)\..\..\include;”

2）链接器。“常规” – “附加库目录”：在 x64 项目添加 “$(OPENCV_DIR)\lib;”。

3）如果使用 MFC，在项目属性的常规-“MFC 的使用”中选择“在静态库中使用 MFC”，否则会产生内存泄漏。

### 2.1.3. 编程

在程序开头包含头文件，引用库文件。例如

```
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#ifdef _DEBUG
#pragma comment(lib,"opencv_world401d.lib")
#else
#pragma comment(lib,"opencv_world401.lib")
```

```
#endif
```

注意 debug 配置下所用的库应该加上后缀 d 才能正常工作。

OpenCV 的类和函数均位于 cv 名字空间。为了简化调用，可在程序开头再加上：

```
using namespace cv;
using namespace std;
```

## 2.2. WinCVMat 库说明

OpenCV 的基本数据结构是 Mat。接口库 WinCVMat 提供了将 Mat 变量数据与文本文件交换和显示 Mat 变量数据的功能。WinCVMat 的成员函数说明见表 1。

表 1 WinCVMat 的 C++函数说明

| 函数类别 | 函数原型 | 说明 |
|---|---|---|
| 文件IO | int SaveMatToText(const char* filename, const cv::Mat& mat) | **功能：**将二维多通道数组按照行保存为文本文件。这样便于在 MATLAB 中进一步分析。<br>**输入：**filename - 文本文件名。mat - 源数组。<br>**输出：**数据写入文本文件。<br>**返回值：**文本行数。<br>**说明：**文本文件的一行为数组的一行，一行数据用逗号分隔，格式如下：第一列通道 1，第一列通道 2，...，第一列通道 K，第二列通道 1，...，第 N 列通道 K。 |
| | int LoadMatFromText(const char* filename, cv::Mat& mat) | **功能：**从文本文件读取二维多通道数组数据。文件可来源于 MATLAB 命令：save '文件名' 变量名 -ascii -double。<br>**输入：**filename - 文本文件名。mat - 获取数据的数组，其行数、列数、通道数规定了文本文件的格式。<br>**输出：**mat - 获取从文本文件读取的数据。数据不足则填 0。<br>**返回值：**文本行数。<br>**说明：**文本文件的一行为数组的一行，一行数据用逗号、分号、“\|”或空白分隔，格式如下：第一列通道 1，第一列通道 2，...，第一列通道 K，第二列通道 1，...，第 N 列通道 K。 |
| 显示函数 | void ShowCVMat(HWND hwnd, const cv::Mat& img, double minVal, double maxVal, int TopDown); | **功能：**在指定窗口的整个可视区域显示整个二维矩阵（矩阵图像保持图像宽高比）。仅支持 1~4 通道数组。<br>**输入：**hwnd -用于显示矩阵的窗口。mat -二维矩阵数据。minVal, maxVal -矩阵元素的下界和上界，超过下界的数值转化为最小强度值，超过上界的数值转化为最大强度值。TopDown -矩阵图像是否自上而下。1 表示矩阵第一行对应图像顶部，0 表示矩阵第一行对应图像底部。<br>**说明：**频繁显示应使用数据类型为 8bit（即 CV_8U 或 CV_8S）的矩阵（不需要数据转换，不需要 minVal 和 maxVal）。 |

| | | |
|---|---|---|
| | void ShowCVMat2(HWND hwnd, const RECT& rectDest, const cv::Mat& img, double minVal, double maxVal, int TopDown); | **功能：**在指定窗口的指定矩形显示整个二维矩阵（矩阵图像保持图像宽高比）。仅支持 1~4 通道数组。<br>**输入：**hwnd -用于显示矩阵的窗口。rectWin -窗口中用于显示的区域。mat -二维矩阵数据。<br>minVal, maxVal -矩阵元素的下界和上界，超过下界的数值转化为最小强度值，超过上界的数值转化为最大强度值。<br>TopDown -矩阵图像是否自上而下。1 表示矩阵第一行对应图像顶部，0 表示矩阵第一行对应图像底部。<br>**说明：**频繁显示应使用数据类型为 8bit（即 CV_8U 或 CV_8S）的矩阵（不需要数据转换，不需要 minVal 和 maxVal）。 |
| | void ShowCVMatEx(HWND hwnd, const RECT& rectWin, const cv::Mat& img, const RECT& rectMat, double minVal, double maxVal, int mirror, DWORD dwRop = SRCCOPY); | **功能：**在指定窗口的指定矩形显示二维矩阵的指定范围。仅支持 1~4 通道矩阵。矩阵图像自上而下，即矩阵第一行对应图像顶部。<br>**输入：**hwnd -用于显示矩阵的窗口。rectWin -窗口中用于显示的区域。<br>mat -二维矩阵数据。rectMat -矩阵中用于显示的区域（不是图像区域）。<br>minVal, maxVal -矩阵元素的下界和上界，超过下界的数值转化为最小强度值，超过上界的数值转化为最大强度值。<br>mirror -镜像显示：0 显示原图像，1 左右镜像，2 上下镜像，3 上下左右镜像。<br>dwRop -光栅操作码，取值参阅 BitBlt。<br>**说明：**频繁显示应使用数据类型为 8bit（即 CV_8U 或 CV_8S）的矩阵（不需要数据转换，不需要 minVal 和 maxVal）。 |

例 1：

```
Mat src = imread("lena.bmp");
ShowCVMat(m_ctrlPic4, src, 0, 255, 1);
```

显示结果如图 1


图 1 ShowCVMat 效果图

例 2：

```
Mat src = imread("lena.bmp");
int img_height = 100;
int img_width = 100;
CRect rectDest;
```

```
rectDest.SetRect(0, 0, img_width, img_height);
ShowCVMat2(m_ctrlPic4, rectDest, src, 0, 255, 1);
```

显示结果如图 2


图 2 ShowCVMat2 效果图

例 3：

```
Mat src = imread("lena.bmp");
int img_height = 200;
int img_width = 200;
CRect rectSrc,rectDest;
rectSrc.SetRect(10, 10, src.rows-20, src.cols - 20);
rectDest.SetRect(40, 40, img_width, img_height);
ShowCVMatEx(m_ctrlPic4, rectDest, src, rectSrc, 0, 255, 3, SRCAND);
```

显示结果如图 3

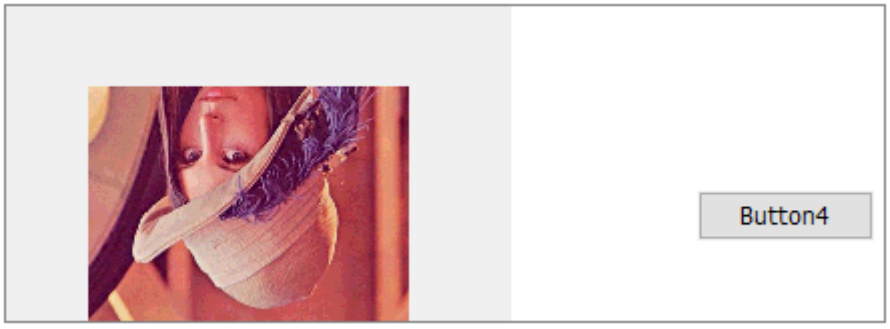


图 3 ShowCVMatEx 效果图

## 2.3. ImageMatrix 库说明

ImageMatrix 调用 API 函数 StretchBlt()实现最基本的图像点阵显示功能。BYTE 型的 pixels 和 Mat 一样都是图像点阵，一般都是三维数组：高、宽、颜色。它们的缓冲区都设置成一维列向量的形式。区别在于 pixels 是 8 bit 数组，而 mat 可以是各种数据类型，但是需要转换为 8 bit 类型才能用 StretchBlt()显示，所以频繁显示应使用 BYTE 类型矩阵。

**1）位图结构**

ImageMatrix 库是专门对“.bmp”格式的图像（位图）进行处理的库。要将位图文

件“.bmp”从磁盘读取到内存，首先要了解其文件结构。位图的文件组成有以下 4 个部分，如图 4。

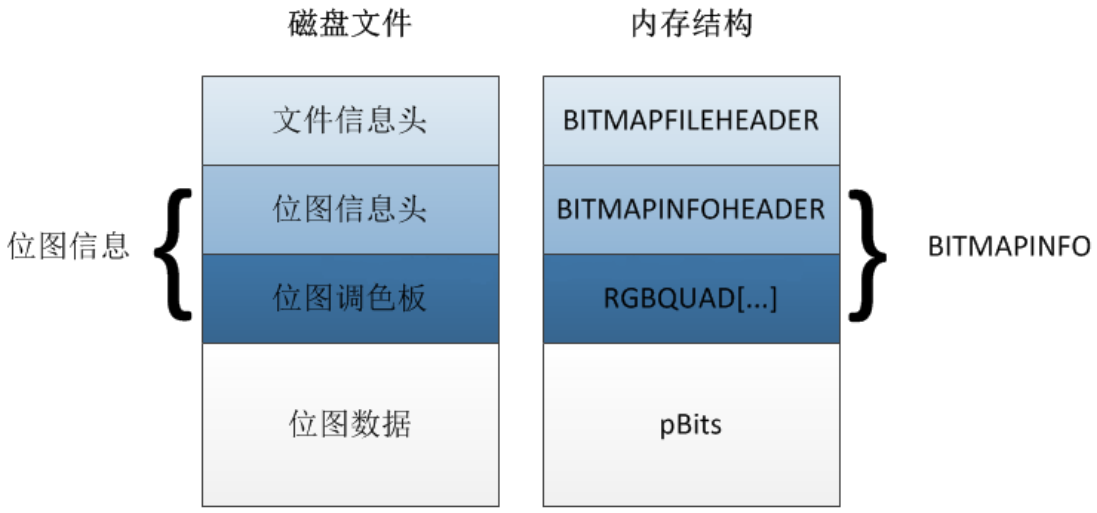


图 4 位图的结构图

a）文件表头，主要包含了文件的类型（必须是 BM），文件的大小（所占用的字节数）和位图的像素矩阵的偏移量。

b）信息表头，包含了两部分内容：位图的相关信息（位图的大小、位深度、位面数、压缩和编码等）和指向 RGB 颜色表（调色板）的指针。

c）RGB 色彩对照表，也就是调色板，不一定会有。16 位及以上直接使用 RGB 通道表示颜色，一般不需要调色板。

d）位图的像素信息矩阵，表示具体的像素。1，4，8 位颜色，保存的是调色板的索引，具体的颜色根据索引在调色板中查找；16 位及其以上不使用调色板，直接使用 RGB 组成像素颜色。

**2）位图矩阵初始化**

图像头缓冲区就是一块内存，用来存储读取到的图像头信息，在调用 ImageMatrix 库时需要初始化。当读取成功时，该指针就指向（该内存中就存储）图像头的信息。设置方式如下：

```
/*图像矩阵的初始化一定要放在全局变量中。因为存储图像矩阵的向量一般都比较大，在控件响应函数中频繁分配和释放那么大的空间很容易导致中断。*/
const int szDib = 5000000;  //szDib的设置灵活可变，比如，要存储一个1024*768的RGB图像，则所需空间为1024*768*3=235 9296，这时，可大致设置szDib=250 0000。
BYTE Dib[szDib];  //图像矩阵缓冲区的设置是用来分配一块内存来存储图像矩阵的像素点信息。
BITMAPINFO* bmi;  //图像头地址
unsigned int m_buffer[sizeof(BITMAPINFOHEADER) + sizeof(RGBQUAD) * 256];  //图像头缓冲区 =40B 图像头 +4*256B 调色板
bmi = (BITMAPINFO*)m_buffer;  //图像头初始化
```

### 3）位图矩阵的读取与显示

ImageMatrix 的成员函数说明见表 2。

表 2 ImageMatrix 的 C++函数说明

| 函数类别 | 函数原型 | 说明 |
|---|---|---|
| 操作函数 | ReadBmpFile(char *filename, BITMAPINFO* bmi, BYTE *Dib, int szDib); | **功能：**读取一个 BMP 文件。仅支持彩色（16~32bit）和黑白（8bit）图像。<br>**输入：**filename - BMP 文件名。bmi - 图像头缓冲区（40B 图像头+4*256B 调色板）。Dib[szDib] - 图像点阵缓冲区。<br>**输出：**bmi - 图像头信息。Dib[szDib] - 图像点阵数据。<br>**返回值：**实际读取的图像点阵字节数，不大于 szDib。-1 表示读取失败，0 表示只读取图像头。<br>**说明：**图像的文件头格式为：BITMAPINFOHEADER 结构（40B）+调色板（4*256B）。<br>8bit 图像的调色板有 256 色，每种颜色是一个 4 字节的 RGBQUAD 结构。16~32bit 图像没有调色板。<br>如果 Dib 为 NULL，则只读取图像头 bi[szBI]和图像点阵字节数 pixelBytes。如果 Dib[szDib]空间不足则点阵不完整。 |
| | ShowBmp(HWND hwnd, const BITMAPINFO* bmi, const BYTE* pixels); | **功能：**在指定窗口的整个可视区域显示整个图像（保持图像宽高比）。<br>**输入：**hwnd - 用于显示图像的窗口。bmi - 图像头结构。pixels - 图像点阵。 |
| | ShowBmp2(HWND hwnd, const RECT& rectDest, const BITMAPINFO* bmi, const BYTE* pixels); | **功能：**在指定窗口的指定矩形显示整个图像（保持图像宽高比）。<br>**输入：**hwnd -用于显示图像的窗口。rectDest -窗口中用于显示的区域。bmi -图像头结构。pixels -图像点阵。 |
| | ShowBmpEx(HWND hwnd, const RECT& rectDest, const BITMAPINFO* bmi, const BYTE* pixels, const RECT& rectSrc, int mirror, DWORD dwRop = SRCCOPY); | **功能：**在指定窗口的指定矩形显示图像的指定范围。<br>**输入：**hwnd -用于显示图像的窗口。rectDest -窗口中用于显示的区域。<br>bmi -图像头结构。pixels -图像点阵。rectSrc -图像中用于显示的区域。<br>mirror -镜像显示：0 显示原图像，1 左右镜像，2 上下镜像，3 上下左右镜像。<br>dwRop -光栅操作码，取值参阅 BitBlt。 |

| | | |
|---|---|---|
| | ReadByteMatrixFromText(char *filename, BYTE *mat, int width, int height); | **功能：**从文本文件读取二维矩阵。文本文件的一行为矩阵的行，一行数据用逗号、分号、“\|”或空白分隔。<br>**输入：**filename -文本文件名。mat[height][width] -源矩阵缓冲区，数据类型取决于 mat_type。<br>mat_type -数据类型，0~9: double (缺省类型), int, unsigned int, char (8 bit), unsigned char(8 bit), short (16 bit), unsigned short (16 bit), long (32 bit), unsigned long (32 bit), float。<br>**输出：**mat[height][width] -从文本文件读取的矩阵。<br>**返回值：**实际读取的行数，不大于 height。 |
| | FillDibHead(BITMAPINFO* bmi, int chnpp, int width, int height, int TopDown); | **功能：**生成图像头结构 bmi。<br>**输入：**bmi -长度为 1064 字节的图像头结构缓冲区。chnpp、width、height -图像通道数（1、3、4）、宽、高。<br>TopDown -矩阵图像是否自上而下。1 表示图像点阵第一行对应图像顶部，0 表示图像点阵第一行对应图像底部。<br>**输出：**bmi -图像头结构信息。<br>**说明：**bmi 的结构为一个 BITMAPINFOHEADER（40 字节）和 256 个 RGBQUAD 结构（1024 字节）。 |

例 1：

```
const int szDib = 5000000;
BYTE Dib[szDib];
BITMAPINFO* bmi;
unsigned int m_buffer[sizeof(BITMAPINFOHEADER) + sizeof(RGBQUAD) * 256];
bmi = (BITMAPINFO*)m_buffer;
ReadBmpFile("lena.bmp", bmi, Dib, szDib);
ShowBmp(m_ctrlPic4, bmi, Dib);
```

显示结果如图 5


图 5 ShowBmp 效果图

例 2：

```
const int szDib = 5000000;
BYTE Dib[szDib];
BITMAPINFO* bmi;
```

```
unsigned int m_buffer[sizeof(BITMAPINFOHEADER) + sizeof(RGBQUAD) * 256];
bmi = (BITMAPINFO*)m_buffer;
ReadBmpFile("lena.bmp", bmi, Dib, szDib);
int img_height = 200;
int img_width = 200;
CRect rectDest;
rectDest.SetRect(10, 10, img_width, img_height);
ShowBmp2(m_ctrlPic4,  rectDest,  bmi,  Dib);
```

显示结果如图 6


图 6 ShowBmp2 效果图

例 3：

```
const int szDib = 5000000;
BYTE Dib[szDib];
BITMAPINFO* bmi;
unsigned int m_buffer[sizeof(BITMAPINFOHEADER) + sizeof(RGBQUAD) * 256];
bmi = (BITMAPINFO*)m_buffer;
ReadBmpFile("lena.bmp", bmi, Dib, szDib);
int img_height = 200;
int img_width = 200;
CRect rectSrc,rectDest;
rectSrc.SetRect(10, 10, bmi->bmiHeader.biWidth-20, bmi->bmiHeader.biHeight - 20);
rectDest.SetRect(10, 10, img_width, img_height);
ShowBmpEx(m_ctrlPic4,  rectDest,  bmi,  Dib,  rectSrc,  3,  SRCAND);
```

显示结果如图 7


图 7 ShowBmpEx 效果图

d）函数 ReadByteMatrixFromText()为读取文本函数，函数 FillDibHead ()为生成图像头结构函数，因为读取文本没有图像头结构信息，故需自行生成图像头结构，以便后续调用。例 4：

```
const int szDib = 5000000;
BYTE Dib[szDib];
BITMAPINFO* bmi;
unsigned int m_buffer[sizeof(BITMAPINFOHEADER) + sizeof(RGBQUAD) * 256];
bmi = (BITMAPINFO*)m_buffer;
ReadByteMatrixFromText("lena.txt", Dib, 512, 512*3);
FillDibHead(bmi, 3, 512, 512, 0);
ShowBmp(m_ctrlPic4, bmi, Dib);
```

显示结果如图 8


图 8 文本读取并显示效果图

# 3. VideoGrabber说明

## 3.1. 概述

VideoGrabber 是用于管理视频设备和获取视频帧的模块，支持多个进程共享读取视频帧。主要功能包括：1）查询视频设备，2）管理视频设备，3）读取视频帧。

VideoGrabber 对象分为**设备管理对象**和**视频帧获取对象**。一个视频设备可以关联

多个 VideoGrabber 对象，这些对象可以属于不同的进程，其中可以有多个视频帧获取对象，但只有且必须有一个设备管理对象。一个视频设备如果没有关联设备管理对象则无法启动，不能关联视频帧获取对象和获取视频帧；如果关联多个设备管理对象则只有一个能起作用。关联同一个视频设备的所有视频帧获取对象可以同时读取视频帧。

此模块分为 C++版（VideoGrabber_x64.dll）和 C#版（MobileCameraControl.dll 和 VideoGrabber_x64.dll）。

## 3.2. C++编程

用 C++编程时注意设置项目属性，包括引用包含文件 VideoGrabber.hpp 和库文件 VideoGrabber_x64.lib，设置系统路径指向 VideoGrabber_x64.dll。

首先应用程序应调用静态函数 VideoGrabber::SearchVideoDevices()搜索系统安装的视频设备。拔插设备后需要再次调用此函数更新设备列表。调用此函数后就可以调用静态函数 VideoGrabber::GetDeviceTotal()获取视频设备总数 N。视频设备用序号标识，序号从 0 开始，最大值为 N-1。

然后应用程序创建一个 VideoGrabber 对象，作为设备管理对象或视频帧获取对象。每一个视频设备应该有一个设备管理对象（可以在其它进程中），否则不能获取视频帧（视频帧获取对象不能工作）。

**（1）设备管理对象使用方法如下：**

①创建一个 VideoGrabber 对象，调用 Open()关联被管理的视频设备。

②用 Start()启动关联的视频设备。如果需要，用 Stop()停止关联的视频设备。

③调用“设备管理”函数设置关联视频设备的属性、视频格式，获取视频信息。对于视频采集卡，调用 GetCapturerInfo()查看可用输入端口。

④用 GetActivatedDeviceNo()、GetActivatedDeviceName()查看关联的设备编号和名称。

⑤不再使用设备时，调用 Close()或删除对象将关闭关联的视频设备。

**（2）视频帧获取对象使用方法如下：**

①创建一个 VideoGrabber 对象。用 SelectSourceDevice()选择关联的视频源设备。

②定时用 ReadFrame()获取视频帧及其信息。注意设备管理对象用 Start()启动设备后，视频帧获取对象才能用 ReadFrame()获得视频帧。

③可以用 GetSourceDeviceNo()、GetSourceDeviceName()查看关联的视频源设备编号和名称。

VideoGrabber 的成员函数说明见表 3。

表 3 VideoGrabber 的 C++函数说明

| 类别 | 函数原型 | 说明 |
|---|---|---|

| | | |
|---|---|---|
| 设备查询函数 | static int SearchVideoDevices(); | **功能：**搜索系统安装的所有视频设备。拔插设备后需要调用此函数更新设备列表。<br>**返回值：**视频设备总数，0 表示搜索失败或无视频设备。 |
| | static int GetDeviceTotal(); | **功能：**取得视频设备总数。调用 SearchVideoDevices()后有效。<br>**返回值：**视频设备总数。-1 表示未调用 SearchVideoDevices()。0 表示无视频设备。 |
| | static char* GetDeviceName(int devNo); | **功能：**取得视频设备名称。调用 SearchVideoDevices()后有效。<br>**输入：**DevNo - 设备序号（从 0 开始）。<br>**返回值：**设备名称字符串指针。NULL 表示 DevNo 大于设备总数。 |
| 设备管理函数 | int Open(int DevNo, int VideoPinType = -1); | **功能：**设备管理对象关联指定编号的视频设备。之前关联的视频设备被停止视频流并断开。<br>**输入：**DevNo - 选中的设备序号（从 0 开始）。VideoPinType - 选择视频采集卡的输入端口类型。缺省值为 PhysConn_Video_Composite(2)。如果指定的设备非视频采集卡，则忽略此参数。<br>**返回值：**返回 1 表示成功，0 表示失败。<br>**说明：**关联设备后才能启动设备和设置视频格式和属性。 |
| | void Close(); | **功能：**停止视频流，断开关联的视频设备和关联的视频源设备。<br>**说明：**对象结束时自动调用此函数。 |
| | int GetActivatedDeviceNo(); | **功能：**获得关联的视频设备编号（从 0 开始）。返回-1 表示未关联视频设备。 |
| | char* GetActivatedDeviceName(); | **功能：**获得关联的视频设备名称。返回 NULL 表示未关联视频设备。 |
| | int GetRunningState(); | **功能：**功能：查询关联的视频设备的视频流状态。<br>**返回值：**返回 0 表示停止，1 表示暂停，2 表示正常运行，-1 表示未关联视频设备，-2 表示状态获取失败。<br>**说明：**视频设备被其它设备管理对象启动则状态查询失败。 |
| | int Start();<br>int Stop();<br>int Pause(); | **功能：**启动、停止、暂停关联的视频设备的视频流。<br>**说明：**启动的视频设备不能被其它设备管理对象管理。 |
| | void SetVideoDeviceDlg(); | **功能：**打开视频设备属性对话框并设置属性。 |
| | char* GetCapturerInfo(); | **功能：**如果当前关联的设备是视频采集卡，则获得其所有输入端口的信息。<br>**返回值：**视频采集卡的输入端口信息字符串指针。NULL 表示未关联设备或非视频采集卡。 |
| | int GetVideoFormatTotal(); | **功能：**读取视频设备所支持的视频格式数量。<br>**返回值：**支持的视频格式数量。 |
| | char* GetVideoFormatName (int FormatNo); | **功能：**读取指定编号的自定义视频格式名称。格式名称由格式编号（从 1 开始）和格式缩写组成。<br>**输入：**FormatNo - 视频设备所支持的视频格式编号（从 0 开始）。<br>**输出：**无。<br>**返回值：**视频格式名称字符串指针。NULL 表示失败。 |

| | | |
|---|---|---|
| | char* GetVideoFormat(int FormatNo, int& Channel, int& Width, int& Height, double& secPerFrame); | **功能：**读取指定编号的视频格式的尺寸和信息字符串。<br>**输入：**FormatNo -视频设备所支持的视频格式编号（从 0 开始）。<br>**输出：**Channel -颜色通道数。Width -视频宽度。Height -视频高度。secPerFrame -每帧秒数。<br>**返回值：**视频格式信息字符串指针。NULL 表示失败。 |
| | char* GetCurrentVideoFormat (int& Channel, int& Width, int& Height, double& secPerFrame); | **功能：**读取当前视频格式的尺寸和信息字符串。<br>**输入：**无。<br>**输出：**Channel -颜色通道数。Width -视频宽度。Height -视频高度。secPerFrame -每帧秒数。<br>**返回值：**视频格式信息字符串指针。NULL 表示失败。 |
| | int SetVideoFormat(int FormatNo, int& Channel, int& Width, int& Height, double& secPerFrame); | **功能：**设置视频格式为指定编号的视频格式，并获得设置后的视频格式的尺寸。<br>**输入：**FormatNo -视频设备所支持的视频格式编号（从 0 开始）。<br>**输出：**Channel -颜色通道数。Width -视频宽度。Height -视频高度。secPerFrame -每帧秒数。<br>**返回值：**返回 1 表示成功，0 表示失败。<br>**说明：**即使格式设置失败可可以得到当前视频格式的尺寸。 |
| 图像获取函数 | int SelectSourceDevice(int DevNo); | **功能：**视频帧获取对象关联指定的视频设备为视频源。该设备启动后就可以获取图像。可以用 Close()取消关联。<br>**输入：**DevNo -选中的设备序号（从 0 开始）。<br>返回值：返回 1 表示成功，0 表示失败。<br>**说明：**选中的视频源设备必须已经在本进程或其它进程关联一个设备管理对象（调用 Open()），否则本函数调用失败。 |
| | int GetSourceDeviceNo(); | **功能：**获得视频帧来源的设备编号（从 0 开始）。返回-1 表示未选择视频源设备。 |
| | char* GetSourceDeviceName (); | **功能：**获得视频帧来源的设备名称。返回 NULL 表示未选择。 |
| | int ReadFrame(BYTE* Dib, int szDib, int& szFrame, double& secStamp, int& Channel, int& Width, int& Height, double& secPerFrame) | **功能：**从关联的视频源读取一个视频帧。<br>**输入：**Dib[szDib] -帧缓冲区。<br>**输出：**Dib[] -视频帧点阵。szFrame -视频帧帧字节数。secStamp -帧的时戳（秒）。Channel -颜色通道数。Width -视频宽度。Height -视频高度。secPerFrame -每帧秒数。<br>返回值：返回 1 表示成功，0 表示失败。<br>**说明：**<br>1）关联的视频源必须存在一个调用 Start()启动该设备的设备管理对象，ReadFrame()才能获得实时的图像帧。<br>2）如果帧缓冲区空间不足（szDib<szFrame），则不获取视频帧，但仍然可以得到视频信息，包括所需字节数 szFrame。<br>3）Channel 是视频设备输出的颜色通道数，而 Dib 中获取的视频帧的颜色通道数总是 3（24bit）。 |

具体调用程序为：

```
VideoGrabber grab;
int snBlock = 1;
const int szBlock = 6000 * 4000 * 3;
```

```
byte buf[szBlock];
int imgWidth = 100, imgHeight = 100, imgChannel = 3, szFrame;
double secStamp, secPerFrame = 33;
BOOL CVisionAppProcessingDlg::OnInitDialog()
{
	// TODO:  在此添加额外的初始化代码
	grab.SearchVideoDevices();
	m_ctrlDeviceList.Clear();
	for (int i = 0; i < grab.GetDeviceTotal(); ++i)
		{
			m_ctrlDeviceList.InsertString(i, CString(grab.GetDeviceName(i)));
		}
	m_ctrlDeviceList.SetCurSel(0);
	grab.SelectSourceDevice(0);
}
void CVisionAppProcessingDlg::OnCbnSelchangeDevicelist()
{
	int DevNo = m_ctrlDeviceList.GetCurSel();
	int ret = grab.SelectSourceDevice(DevNo);
}
void CVisionAppProcessingDlg::OnTimer(UINT_PTR nIDEvent)
{
	if (nIDEvent == 1)
	{
		grab.ReadFrame(buf, sizeof(buf), szFrame, secStamp, imgChannel, imgWidth, imgHeight,
secPerFrame);
	}
CDialog::OnTimer(nIDEvent);
}
```

显示结果如图 9

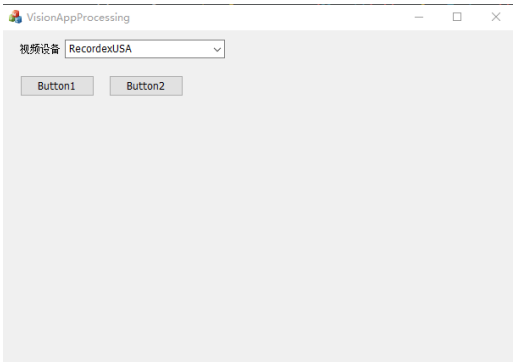


图 9 VideoGrabber 调用效果图

## 3.3. C#编程

用 C#编程时注意引用 MobileCameraControl.dll 和 VideoGrabber_x64.dll，命名空间为 Vision。

MobileCameraControl.dll 包含以下三个类：

1）VideoGrabber：视频设备类，实现视频设备访问的基本功能，包括查询视频设备、管理视频设备和获取视频帧。该类可以单独使用，但它不含任何界面，因此用户需要设计界面，或者改用下面两个类。

2）VideoSetupControl：用于管理视频设备和获取视频帧的用户控件。该类可以单独使用，但需要设计显示界面，或用 CamAttitudeControl 显示视频。

3）CamAttitudeControl：用于显示视频帧和相机姿态的用户控件。该类仅用于显示，不获取视频帧。用户可以用 VideoGrabber 或 VideoSetupControl 获取视频帧。

### 3.3.1. VideoGrabber

VideoGrabber 提供编写视频应用程序所需的完整功能，但不包含用户界面，因此用户需要设计界面。

首先应用程序应调用静态函数 VideoGrabber.SearchVideoDevices()搜索系统安装的视频设备。拔插设备后需要再次调用此函数更新设备列表。调用此函数后就可以调用静态函数 VideoGrabber.GetDeviceTotal()获取视频设备的总数 N。视频设备用序号标识，序号从 0 开始，最大值为 N-1。

然后应用程序创建一个 VideoGrabber 对象，作为设备管理对象或视频帧获取对象。每一个视频设备应该有一个设备管理对象（可以在其它进程中），否则不能获取视频帧（视频帧获取对象不能工作）。

**（1）设备管理对象使用方法如下：**

①创建一个 VideoGrabber 对象，调用 Open()关联被管理的视频设备。

②用 Start()启动关联的视频设备。如果需要，用 Stop()停止关联的视频设备。

③调用“设备管理”函数设置关联视频设备的属性、视频格式，获取视频信息。对于视频采集卡，调用 GetCapturerInfo()查看可用输入端口。

④用 GetActivatedDeviceNo()、GetActivatedDeviceName()查看关联的设备编号和名称。

⑤不再使用设备时，调用 Close()或删除对象将关闭关联的视频设备。

**（2）视频帧获取对象使用方法如下：**

①创建一个 VideoGrabber 对象。用 SelectSourceDevice()选择关联的视频源设备。

②定时用 ReadFrame()获取视频帧及其信息。将返回的 Bitmap 对象赋值给窗体中的 PictureBox 控件的 Image 属性可以显示图像。注意设备管理对象用 Start()启动设备后，视频帧获取对象才能用 ReadFrame()获得视频帧。

③可以用 GetSourceDeviceNo()、GetSourceDeviceName()查看关联的视频源设备编号和名称。

VideoGrabber 的成员函数说明见表 4。

表 4 VideoGrabber 的 C#函数说明

| 函数类别 | 函数原型 | 说明 |
|---|---|---|
| 设备查询函数 | static int SearchVideoDevices (); | **功能：**搜索系统安装的所有视频设备。拔插设备后需要调用此函数更新设备列表。<br>**返回值：**视频设备总数，0 表示搜索失败或无视频设备。 |
| | static int GetDeviceTotal(); | **功能：**取得视频设备总数。调用 SearchVideoDevices()后有效。<br>**返回值：**视频设备总数。-1 表示未调用 SearchVideoDevices()。0 表示无视频设备。 |
| | static string GetDeviceName (int devNo); | **功能：**取得视频设备名称。调用 SearchVideoDevices()后有效。<br>**输入：**DevNo -设备序号（从 0 开始）。<br>**返回值：**设备名称字符串指针。NULL 表示 DevNo 大于设备总数。 |
| | static int FillDeviceList (ComboBox list)<br>static int FillDeviceList (ListBox list) | **功能：**用视频设备名称填充列表框。<br>**输入：**list -列表框。<br>**返回值：**视频设备总数。-1 表示未调用 SearchVideoDevices()。0 表示无视频设备。 |
| 设备管理函数 | int Open(int DevNo, int VideoPinType = -1); | **功能：**设备管理对象关联指定编号的视频设备。之前关联的视频设备被停止视频流并断开。<br>**输入：**DevNo -选中的设备序号（从 0 开始）。VideoPinType -选择视频采集卡的输入端口类型。缺省值为 PhysConn_Video_Composite(2)。如果指定的设备非视频采集卡，则忽略此参数。<br>**返回值：**返回 1 表示成功，0 表示失败。<br>**说明：**关联设备后才能启动设备和设置视频格式、属性。 |
| | void Close(); | **功能：**停止视频流，断开关联的视频设备和关联的视频源设备。<br>**说明：**对象结束时自动调用此函数。 |
| | int GetActivatedDeviceNo(); | **功能：**获得关联的视频设备编号（从 0 开始）。返回-1 表示未关联视频设备。 |
| | int GetRunningState(); | **功能：**功能：查询关联的视频设备的视频流状态。<br>**返回值：**返回 0 表示停止，1 表示暂停，2 表示正常运行，-1 表示未关联视频设备，-2 表示状态获取失败。<br>**说明：**视频设备被其它设备管理对象启动则状态查询失败。 |
| | int Start();<br>int Stop();<br>int Pause(); | **功能：**启动、停止、暂停关联的视频设备的视频流。<br>**说明：**启动的视频设备不能被其它设备管理对象管理。 |
| | void SetVideoDeviceDlg(); | **功能：**打开视频设备属性对话框并设置属性。 |
| | string GetCapturerInfo(); | **功能：**如果当前关联的设备是视频采集卡，则获得其所有输入端口的信息。<br>**返回值：**视频采集卡的输入端口信息字符串指针。NULL 表示未关联设备或非视频采集卡。 |
| | int GetVideoFormatTotal(); | **功能：**读取视频设备所支持的视频格式数量。<br>**返回值：**支持的视频格式数量。 |

| | | |
|---|---|---|
| | string GetVideoFormatName (int FormatNo); | **功能：**读取指定编号的自定义视频格式名称。格式名称由格式编号（从 1 开始）和格式缩写组成。<br>**输入：** FormatNo -视频设备所支持的视频格式编号（从 0 开始）。<br>**输出：** 无。<br>**返回值：** 视频格式名称字符串指针。NULL 表示失败。 |
| | string GetVideoFormat(int FormatNo, out int Channel, out int Width, out int Height, out double secPerFrame) | **功能：** 读取指定编号的视频格式的尺寸和信息字符串。<br>**输入：** FormatNo -视频设备所支持的视频格式编号（从 0 开始）。<br>**输出：** Channel -颜色通道数。Width -视频宽度。Height -视频高度。secPerFrame -每帧秒数。<br>**返回值：** 视频格式信息字符串指针。NULL 表示失败。 |
| | int SetVideoFormat(int FormatNo, out int Channel, out int Width, out int Height, out double secPerFrame) | **功能：** 设置视频格式为指定编号的视频格式，并获得设置后的视频格式的尺寸。<br>**输入：** FormatNo -视频设备所支持的视频格式编号（从 0 开始）。<br>**输出：** Channel -颜色通道数。Width -视频宽度。Height -视频高度。secPerFrame -每帧秒数。<br>**返回值：** 返回 1 表示成功，0 表示失败。<br>**说明：** 格式设置失败也可以得到当前视频格式的尺寸。 |
| | string GetCurrentVideoFormat (out int Channel, out int Width, out int Height, out double secPerFrame) | **功能：** 读取当前视频格式的尺寸和信息字符串。<br>**输入：** 无。<br>**输出：** Channel -颜色通道数。Width -视频宽度。Height -视频高度。secPerFrame -每帧秒数。<br>**返回值：** 视频格式信息字符串指针。NULL 表示失败。 |
| 图像获取函数 | int SelectSourceDevice(int DevNo); | **功能：** 视频帧获取对象关联指定的视频设备为视频源。该设备启动后就可以获取图像。可以用 Close()取消关联。<br>**输入：** DevNo - 选中的设备序号（从 0 开始）。<br>**返回值：** 返回 1 表示成功，0 表示失败。<br>**说明：** 选中的视频源设备必须已经在本进程或其它进程关联一个设备管理对象（调用 Open()），否则本函数调用失败。 |
| | int GetSourceDeviceNo(); | **功能：** 获得视频帧来源的设备编号（从 0 开始）。返回-1 表示未选择视频源设备。 |
| | Bitmap ReadFrame(out int szFrame, out double secStamp, out double secPerFrame) | **功能：** 从选定的视频源获取一个视频帧及其信息。视频帧的颜色通道数总是 3（Format24bppRgb）。<br>**输出：** szFrame -帧字节数。secStamp -帧的时戳（秒）。secPerFrame -每帧秒数。<br>**返回值：** 由视频帧构造的 Bitmap 对象。null 表示失败。 |

### 3.3.2. VideoSetupControl

VideoSetupControl 主要用于管理视频设备，它为系统每个视频设备创建一个设备管理对象，保证每个设备都可以关联视频帧获取对象。它也可以获取视频帧，但不提供显示界面，应用程序需要设计显示界面。VideoSetupControl 的界面见图 10。

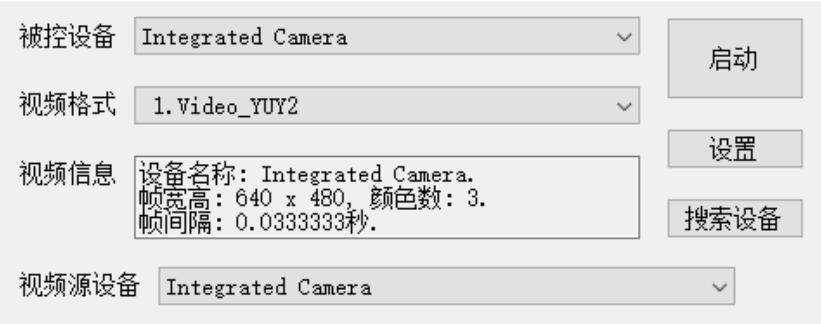


图 10 VideoSetupControl 界面

VideoSetupControl 使用步骤如下：

①在窗体中添加一个 VideoSetupControl 控件（例如 vSetup1）。

②在主窗体的 InitializeComponent()后调用 **InitializeControl**()初始化控件。

③如果需要显示视频，则

- 在窗体中添加一个 PictureBox 控件（例如 pictureBox1）和一个定时器（例如 timer1），设置 **pictureBox1.SizeMode = PictureBoxSizeMode.Zoom**，设置定时器间隔并启动 **timer1.Interval = 100; timer1.Start();**。
- 在定时器事件中，调用 **vSetup1.sourceDevice.ReadFrame()**获取视频帧（Bitmap 对象，记为 frame），然后赋值 **pictureBox1.Image=frame**。
- 在“视频源”列表中可以选择视频源设备。在“被控设备”列表中选择视频源设备并点击“启动”才可以看到视频。

④界面操作：在“被控设备”列表选择设备，点击“启动/停止”开始和停止设备工作。视频设备启动后能输出视频帧。如果插入了新设备，点击“重新搜索视频设备”刷新设备列表。

可以在程序中对设备进行操作。可访问的属性和方法如下。

| 属性和方法 | 说明 |
|---|---|
| VideoGrabber[] videoDevices | 设备管理对象数组 |
| VideoGrabber sourceDevice | 视频帧获取对象 |
| int device_total | 视频设备总数 |
| int currentDevice_index | 当前设备索引号 |
| int frame_channel, frame_width, frame_height, secPerFrame | 视频帧信息-通道数、宽度、高度、帧间隔时间（帧率倒数） |
| void InitializeControl() | 初始化对象，搜索安装的视频设备。等效点击“重新搜索视频设备”。 |
| int SelectCurrentDevice(int index) | 选择当前设备。等效在“被控设备”列表选择设备。 |
| int SelectCurrentDeviceFormat(int index) | 选择当前设备的视频格式。等效在“视频格式”列表选择格式。 |
| int SelectSourceDevice(int index) | 选择视频源设备。在“视频源设备”列表选择设备。 |
| void CurrentDeviceStart() | 启动当前设备视频流。等效点击“启动”。 |

| | |
|---|---|
| void CurrentDeviceStop() | 停止当前设备视频流。等效点击“停止”。 |
| int CurrentDeviceRunningState() | 获得当前设备视频流状态，返回 0 表示停止，1 表示暂停，2 表示正常运行，-1 表示未关联视频设备，-2 表示状态获取失败。 |
| void CurrentDeviceSetup() | 打开当前设备设置对话框。等效点击“设置”。 |

### 3.3.3. CamAttitudeControl

CamAttitudeControl 仅提供显示窗，见图 11，不获取视频帧。应用程序可以用 VideoGrabber 或 VideoSetupControl 获取视频帧。


图 11 CamAttitudeControl 界面

CamAttitudeControl 使用步骤如下：

①在窗体中添加一个 CamAttitudeControl 控件（例如 Cam1）作为显示窗，调整其大小。

②设置 Cam1 的属性：相机焦距 FocalLen、传感器大小 FilmWidth 和 FilmHeight，用于计算 FOV：$\alpha = 2\tan^{-1}(d/2f)$，其中 $f$ 是焦距，$d$ 是传感器大小。横向 FOV 是图像左右对应的方向角，纵向 FOV 是图像上下对应的俯仰角。缺省为 Cannon 5D Mark III 参数：50mm、35.8mm×23.9mm。

③定时获取视频帧 frame 和相机姿态，然后调用 Cam1.ShowFrameAndAttitude()显示视频帧和相机姿态，其中窗口垂直两侧显示俯仰角，上下两行显示朝向角；如果仅显示视频帧 frame，则执行 Cam1.CamWindow.Image=frame。

④可以调整外观如下：ScaleMargin 是刻度的宽度，NumberMargin 是数字占用的宽度和高度，degStep 是刻度间的度数差，sizeFont 是数字的大小。

# 4. DataSharer说明

DataSharer 通过共享数据块实现进程间数据共享。共享数据块由名称和序号标识，名称加上前缀"Global\\"表示允许跨网络访问。

此模块分为 C++ 版（DataSharer_x64.dll）和 C# 版（DataSharerCs.dll 和 DataSharer_x64.dll）。使用方法如下：

1）创建 DataSharer 对象。

2）调用 OpenSharedBlock()创建共享数据块，设置名称、序号和大小。数据块大小应能容纳数据信息和数据内容。用 GetBlockName()和 GetBlockSN()可以查询数据块名称和序号。

3）调用 WriteData()写入一般数据及其信息，调用 ReadData()读取一般数据及其信息。

4）调用 WriteImage()写入图像数据及其信息，调用 ReadImage()读取图像数据及其信息。

共享数据块创建后不能改变大小。如果需要增大则调用 OpenSharedBlock()并改变数据块名称或序号。

用 C++编程时注意设置项目属性，包括引用包含文件 DataSharer.hpp 和库文件 DataSharer_x64.lib，设置系统路径指向 DataSharer_x64.dll。

用 C#编程时注意引用 DataSharerCs.dll 和 DataSharer_x64.dll。前者包含 C#类 DataSharer，命名空间为 Vision。

DataSharer 的 C++成员函数和 C#成员函数的说明分别见表 5 和表 6。

表 5 DataSharer 的 C++函数说明

| 函数原型 | 说明 |
| --- | --- |
| int OpenSharedBlock(const char* BlockName, INT32 snBlock, INT32 szBlock) | **功能：**创建共享数据块，并释放之前创建的共享数据块。<br>**输入：**BlockName -数据块名称，最长 255 字节。snBlock -数据块序号。szBlock -数据块字节数，应能容纳数据信息和数据内容。<br>**返回值：**成功则返回共享内存大小，否则返回 0。<br>**说明：**数据块名称和序号用于标识共享数据块。如果存在同名数据块并且字节数小于 szBlock 则失败。 |
| int WriteData(BYTE* infoSrc, INT32 szInfo, BYTE* dataSrc, INT32 szData, INT32 type); | **功能：**将数据及其信息写入共享数据块。<br>**输入：**infoSrc[szInfo] -被写入的数据信息。dataSrc[szData] -被写入的数据内容。type -数据类型，0 是一般数据，1 是图像。<br>**返回值：**写入的字节数（包括信息和数据），0 表示写入失败。<br>**说明：**先写信息再写数据。infoSrc 为 NULL 或 szInfo 为 0 则无数据信息，只写入数据。dataSrc 为 NULL 或 szData 为 0 则写入失败。 |
| int ReadData(BYTE* dest, INT32 szDest, INT32& byteInfo, INT32& byteData, INT32& type, double& secStamp, INT32& updated); | **功能：**从共享数据块读取数据及其信息。<br>**输入：**dest[szDest] -用于保存数据及其信息的缓冲区。<br>**输出：**dest - 读出的数据及其信息，信息在前数据在后。byteInfo -数据信息的字节数。byteData -数据内容的字节数。数据内容紧跟数据信息。type -数据类型。0 是一般数据，1 是图像。secStamp -数据更新的时戳（秒）。updated - 1 表示上次调用 Read 之后数据已经更新。<br>**返回值：**读出的字节数（包括信息和数据），0 表示读取失败。<br>**说明：**dest 为 NULL 或 szDest 为 0 则仅读取数据块信息，返回值是可读取的数据字节数。 |

| | |
|---|---|
| int WriteImage(BYTE* imgSrc, INT32 szImgSrc, INT32 Width, INT32 Height, INT32 Channel, double secPerFrame); | **功能：**将图像及其信息写入共享数据块。<br>**输入：**imgSrc[szSrcImg] -被写入的图像。Width，Height，Channel -图像的宽、高和颜色通道。secPerFrame -传送图像序列时两幅图像之间的时间间隔（秒）。<br>**返回值：**图像数据字节数，0 表示写入失败。<br>**说明：**图像数据字节数为 Channel×Width×Height，其中 Channel×Width 应为 4 的倍数。 |
| int ReadImage(BYTE* imgDest, INT32 szImgDest, InfoImage& infoImg, double& secStamp, INT32& updated); | **功能：**从共享数据块读取图像及其信息。<br>**输入：**imgDest[szImgDest] - 用于保存图像的缓冲区。<br>**输出：**imgDest - 读出的图像数据。infoImg -图像信息。secStamp -数据更新的时戳（秒）。updated - 1 表示上次调用 Read 之后数据已经更新。<br>**返回值：**读出的图像数据字节数，0 表示失败。<br>**说明：**imgDest 为 NULL 或 szImgDest 为 0 则仅读取图像信息，返回值是可读取的字节数。headerData.typeData 应为 1，否则读取失败。 |
| const char* GetBlockName();<br>int GetBlockSN();<br>int GetBlockBytes(); | **功能：**读取共享数据块的名称、序号和大小。NULL 和-1 表示未成功调用 Openxxx()。 |

表 6 DataSharer 的 C#函数说明

| 函数原型 | 说明 |
|---|---|
| int OpenSharedBlock(string BlockName, int snBlock, int szBlock) | **功能：**创建共享数据块，并释放之前创建的共享数据块。<br>**输入：**BlockName -数据块名称，最长 255 字节。snBlock -数据块序号。szBlock -数据块字节数，应能容纳数据信息和数据内容。<br>**返回值：**成功则返回共享内存大小，否则返回 0。<br>**说明：**数据块名称和序号用于标识共享数据块。如果存在同名数据块并且字节数小于 szBlock 则失败。 |
| int WriteData(IntPtr infoSrc, int szInfo, IntPtr dataSrc, int szData, int type) | **功能：**将数据及其信息写入共享数据块。<br>**输入：**infoSrc，szInfo -被写入的数据信息及其字节数。dataSrc，szData -被写入的数据内容及其字节数。type -数据类型，0 是一般数据，1 是图像。<br>**返回值：**写入的字节数（包括信息和数据），0 表示写入失败。<br>**说明：**先写信息再写数据。infoSrc 为 IntPtr.Zero 或 szInfo 为 0 则无数据信息，只写入数据。dataSrc 为 IntPtr.Zero 或 szData 为 0 则写入失败。 |

| | |
|---|---|
| int ReadData(IntPtr dest, int szDest, out int byteInfo, out int byteData, out int type, out double secStamp, out Int32 updated) | **功能：**从共享数据块读取数据及其信息。<br>**输入：**dest，szDest -用于保存数据及其信息的缓冲区及其大小，可以用 Marshal 的 AllocHGlobal()或 AllocCoTaskMem()方法获取。<br>**输出：**dest -读出的数据及其信息，信息在前数据在后。用 Marshal 的方法可以访问。用 IntPtr.Add(dest,byteInfo)可以获得数据指针。byteInfo -数据信息的字节数。byteData -数据内容的字节数。数据内容紧跟数据信息。type -数据类型，0 是一般数据，1 是图像。secStamp -数据更新的时戳（秒）。updated - 1 表示上次调用 Read 之后数据已经更新。<br>**返回值：**读出的字节数（包括信息和数据），0 表示读取失败。<br>**说明：**dest 为 IntPtr.Zero 或 szDest 为 0 则仅读取数据块信息，返回值是可读取的数据字节数。 |
| int WriteImage(Bitmap imgSrc, double secPerFrame) | **功能：**将图像及其信息写入共享数据块。<br>**输入：**imgSrc -被写入的图像。secPerFrame -传送图像序列时两幅图像之间的时间间隔（秒）。<br>**返回值：**图像数据字节数，0 表示写入失败。<br>**说明：**图像数据字节数为 Channel×Width×Height，其中 Channel×Width 应为 4 的倍数。 |
| Bitmap ReadImage(out double secPerFrame, out double secStamp, out Int32 updated) | **功能：**从共享数据块读取图像及其信息。<br>**输出：**byteBlock -共享数据块大小。secPerFrame -两次写入图像的时间间隔。secStamp -数据时戳。updated - 1 表示上次调用 Read 之后数据已经更新。<br>**返回值：**读出的图像，null 表示失败。 |
| public String GetBlockName()<br>int GetBlockSN()<br>int GetBlockBytes() | **功能：**读取共享数据块的名称、序号和大小。NULL 和-1 表示未成功调用 Openxxx()。 |
| static PixelFormat ChannelToPixelFormat(int channel) | **功能：**将颜色通道数转换为 PixelFormat 类型。 |

具体调用程序为：

```
//C++程序
#include "DataSharer.hpp"
#pragma comment(lib,"DataSharer_x64.lib")
DataSharer shared;
char* SharedBlockName = "VisionProcessingApp";
int snBlock = 1;
const int szBlock = 6000 * 4000 * 3;
byte buf[szBlock];
int imgWidth = 100, imgHeight = 100, imgChannel = 3, szFrame;
double secStamp, secPerFrame = 33;
```

```
BOOL CVisionAppProcessingDlg::OnInitDialog()
{
	// TODO: 在此添加额外的初始化代码
	shared.OpenSharedBlock(SharedBlockName, snBlock, szBlock);
}
void CVisionAppProcessingDlg::OnTimer(UINT_PTR nIDEvent)
{
	Mat matShow;
	shared.WriteImage(matShow.data, matShow.total()*matShow.elemSize(), matShow.cols,
matShow.rows, matShow.channels(), secPerFrame);
}
//C#程序
using Vision;
namespace VisionAppForm
{
	public partial class Form1 : Form
	{
		public Form1()
		{
			shared.OpenSharedBlock(SharedBlockName, snBlock, szBlock);
		}
			DataSharer shared = new DataSharer();
			string SharedBlockName = "VisionProcessingApp";
			private void timer1_Tick(object sender, EventArgs e)
			{
				int szFrame, updated;
				double secStamp, secPerFrame;
				Bitmap frame = videoSetupControl1.sourceDevice.ReadFrame(out szFrame, out
		secStamp, out secPerFrame);
				Bitmap imgProcessed = shared.ReadImage(out secPerFrame, out secStamp, out
		updated);
			}
	}
}
```