# 第 5 章

CHAPTER 5

# MFC＋OpenCV 视频采集及播放

6min

OpenCV 4 是目前最流行的计算机视觉处理库之一，是一个开源的计算机视觉库（Open Source Computer Vision Library，OpenCV）。简单来讲，OpenCV 4 可以用来对图像进行处理，而图像处理一般指数字图像处理（Digital Image Processing），通过数学函数和图像变化等手段对二维的数字图像进行分析，获得图像数据的潜在信息，通常包括图像压缩、增强和复原、匹配和识别 3 部分，涵盖噪声去除、分割和特征提取等处理方法和技术。OpenCV 是由一系列 C 语言函数和 C++类构成的，除了支持使用 C/C++语言进行开发以外，还支持 C＃和 Ruby 等编程语言，并提供了 Python、MATLAB 和 Java 等编程语言接口，可以在 Linux、Windows、macOS、Android 和 iOS 等系统上运行。OpenCV 4 的应用非常广泛，包括图像存储容器、图像的读取与显示、视频加载与摄像头调用、图像变换、图像金字塔、图像直方图的绘制、图像的模板匹配、图像卷积、图像的边缘检测、腐蚀与膨胀、形状检测、图像分割、特征点检测与匹配、单目和双目视觉和光流法目标跟踪，以及在机器学习方面的应用等。本章侧重于讲解 OpenCV 4 在视频方面的应用，包括视频采集和播放等。

## 5.1 使用 VS 2015 搭建 OpenCV 4 开发环境

本节内容使用 VS 2015 配置 OpenCV 4 的开发环境，读者也可以选择不同版本的 VS，需要注意的是，不同的 OpenCV 4 版本对应不同的 VS。

### 1. 下载并配置 OpenCV 4

首先下载 OpenCV 4. x，笔者这里选择的版本是 OpenCV-4. 1. 0（读者也可以选择较新的版本），下载网址为 https://opencv. org/releases/。下载之后将文件（opencv-4. 1. 0-vc14_vc15. exe）放到任意目录下即可，如图 5-1 所示。双击该文件，然后单击 Extract 按钮，如图 5-2 所示。此时会弹出 Extracting 窗口并显示进度，如图 5-3 所示。解压后得到下面的 opencv 文件夹，如图 5-4 所示。为了清楚地区分各个版本，可以把刚才解压后的文件夹添加上版本号，笔者这里将文件夹名修改为 opencv-4. 1. 2，如图 5-5 所示。


图 5-1　下载 OpenCV 4


图 5-2　解压 OpenCV 4

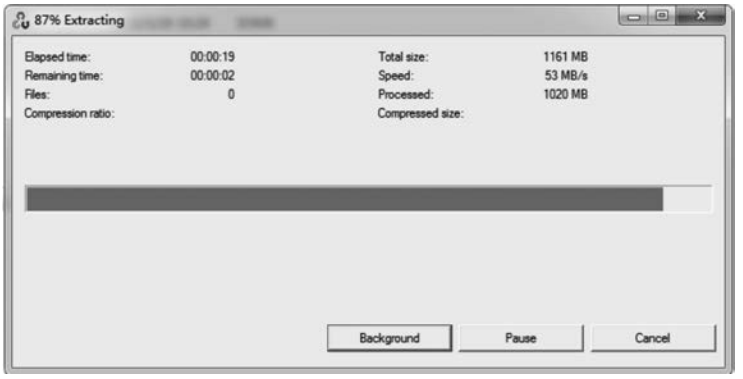


图 5-3　OpenCV 4 安装进度


图 5-4　安装完成后的 OpenCV 4


图 5-5　重命名 OpenCV 4

然后将 OpenCV 4 的运行时动态库文件 opencv_world410.dll 所在的 bin 路径(笔者本机路径为 D:\Program Files\opencv-4.1.2\build\x64\vc14\bin)添加进系统环境变量的 Path 条目中。在桌面上右击“计算机”图标,在弹出的菜单中选择“属性”,然后在弹出的界面中单击“高级系统设置”,然后单击“环境变量”按钮,如图 5-6 所示。此时在弹出的“环境变量”对话中选择 Path 变量,单击“编辑”按钮,在文本框中输入刚才的 bin 路径,最后单击“确定”按钮即可,如图 5-7 所示。

**注意**:官方发布的 OpenCV 根据版本不同而对应不同的 VS 版本,例如 vc14 代表 VS 2015、vc15 代表 VS 2017、vc16 代表 VS 2019,所以读者要根据自己本机的 VS 版本来选择对应的 OpenCV 版本。


图 5-6 修改环境变量

## 2. 配置 VS 2015 项目的 OpenCV 4 开发环境

打开 VS 2015,在起始页单击“新建项目...”,如图 5-8 所示,然后在弹出的“新建项目”对话框中,左侧选择 Visual C++,右侧选择“Win32 控制台应用程序”,输入项目名称和路径,如图 5-9 所示。在弹出的“Win32 应用程序向导”对话框中单击“下一步”按钮,如图 5-10 所示,然后在应用程序类型下选择“控制台应用程序”,在附加选项中可以取消“预编译头”,如图 5-11 所示。


图 5-7 设置 OpenCV 4 的 Path 环境变量

接下来配置 VS 2015 的项目属性,由于官方发布的 OpenCV 4 只编译好了 64 位的库,所


图 5-8　VS 2015 新建项目

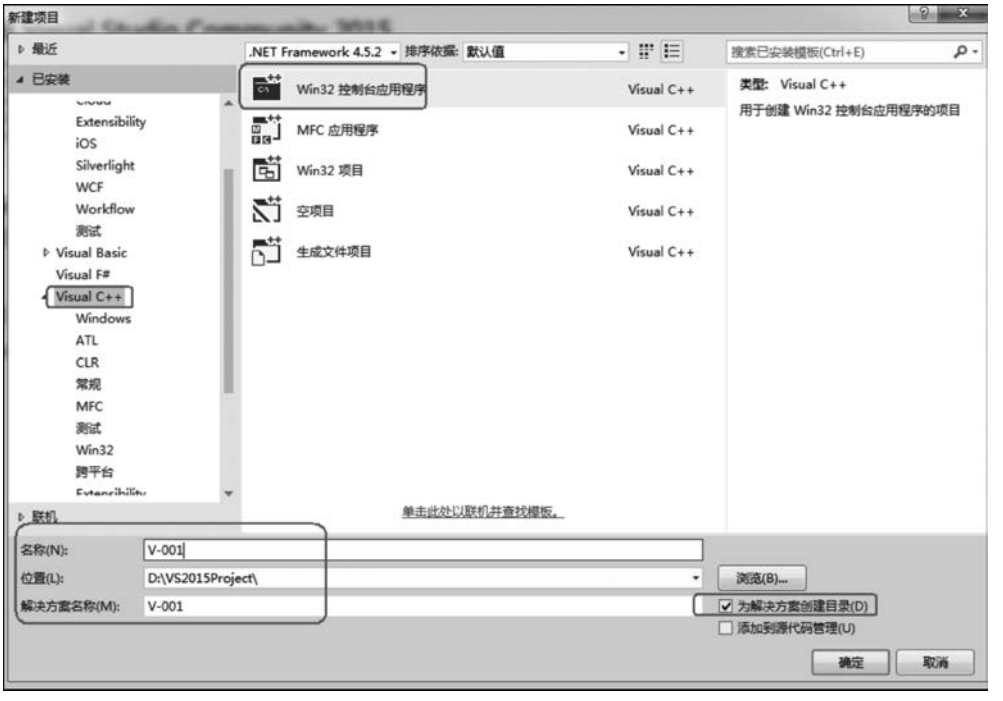


图 5-9　控制台项目的名称与位置


**图 5-10　单击“下一步”按钮**


**图 5-11　取消“预编译头”**

以在 VS 2015 中需要把项目切换到 64 位的开发模式，如图 5-12 所示。右击项目名称，选择“VC++目录→包含目录”，此时单击右侧的“下拉三角”图标，然后单击“<编辑...>”，如图 5-13 所示，然后将 opencv-4.1.2 的头文件路径添加进来即可，如图 5-14 所示，笔者本地的头文件路径如下：

```
D:\Program Files\opencv－4.1.2\build\include\opencv2
D:\Program Files\opencv－4.1.2\build\include
```

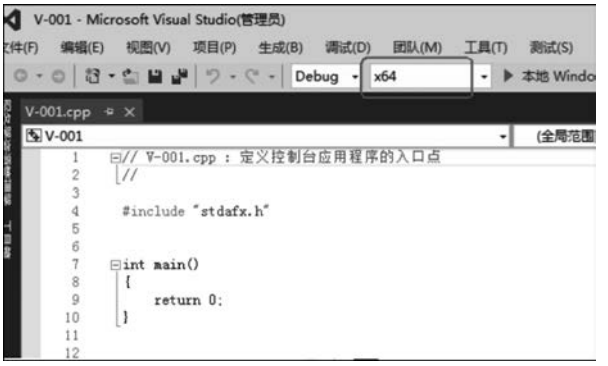


图 5-12　将项目配置为 x64 开发模式


图 5-13　配置项目的“包含目录”

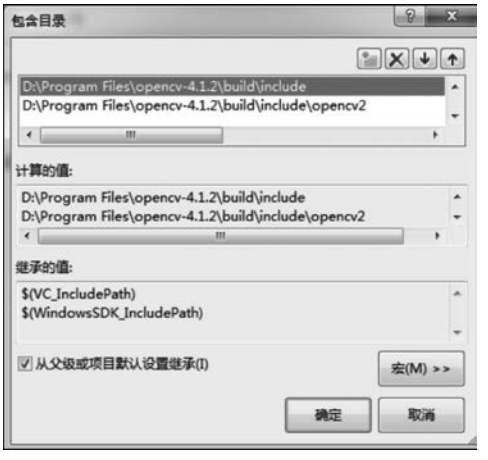


图 5-14　将 OpenCV 的 include 目录添加到“包含目录”

然后添加库目录，右击项目名称，选择“VC++目录→库目录”，此时单击右侧的“下拉三角”图标，然后单击“<编辑...>”，如图 5-15 所示，然后将 opencv-4.1.2 的库文件路径添加进来即可，如图 5-16 所示，笔者本地的库文件路径如下：

```
D:\Program Files\opencv-4.1.2\build\x64\vc14\lib
```

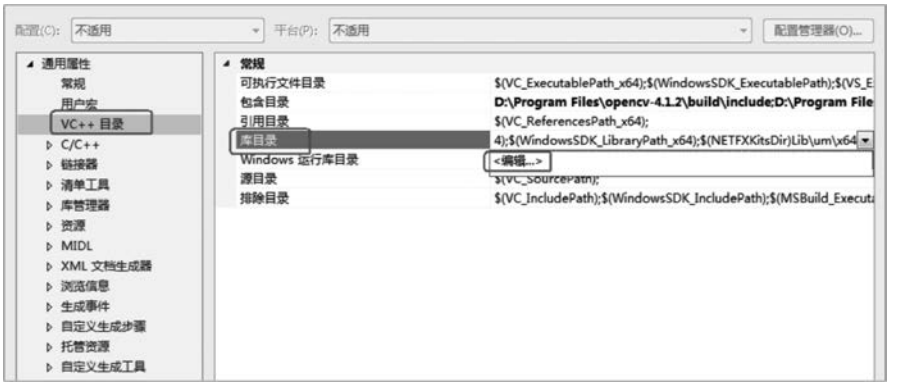


**图 5-15　配置项目的“库目录”**

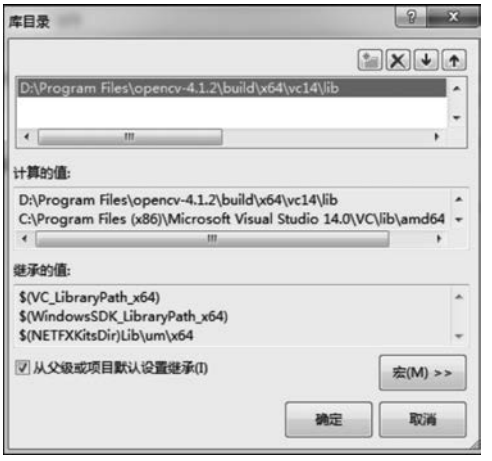


**图 5-16　将 OpenCV 的 lib 目录添加到“库目录”**

最后设置附加依赖项，右击项目名称，然后依次选择“链接器→输入→附加依赖项”，在弹出的对话框中输入 opencv_world412d.lib 即可，如图 5-17 所示。这里的 opencv_world412d.lib 文件实际上就是路径 D:\Program Files\opencv-4.1.2\build\x64\vc14\lib 下面带 d 后缀的 lib 文件，如图 5-18 所示。

**注意**：应将 ax 和 dll 放在一起，否则即使注册后，也无法播放视频。另外，AX 文件名称不要写错了，因为如果输入任意的 AX 文件名，则即便文件不存在，也不会提示注册错误。

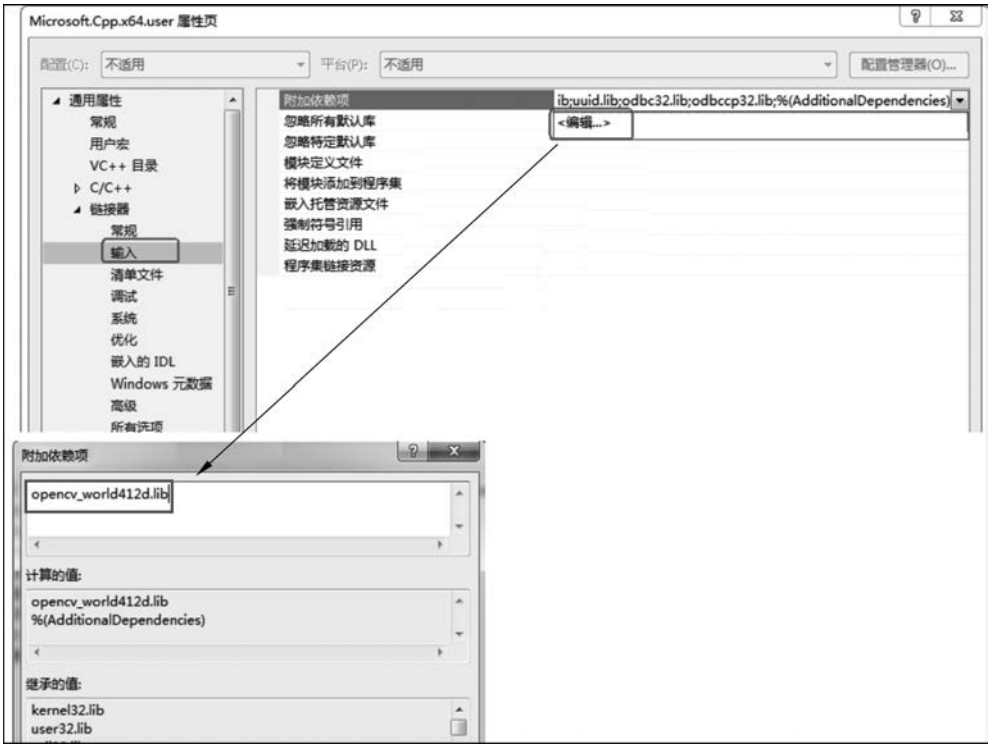


图 5-17　附加依赖项 opencv_world412d.lib

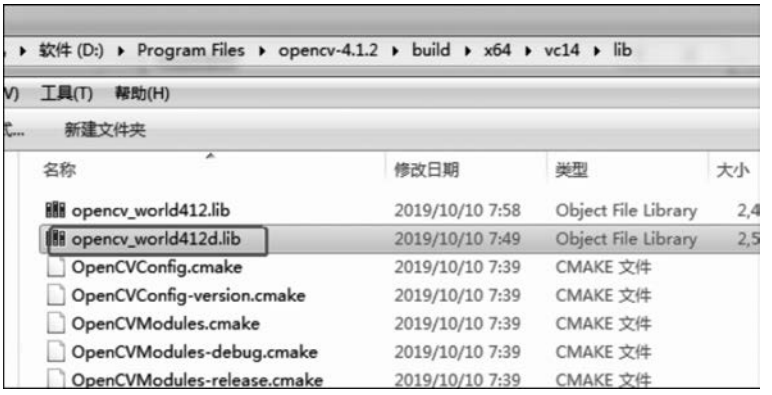


图 5-18　本机 OpenCV 4 的 lib 文件

## 5.2　OpenCV 显示摄像头及磨皮美颜

使用 OpenCV 4 可以很方便地显示图像和摄像头，本节重点讲解 OpenCV 4 显示摄像头及各种特效。

**注意**：本节案例的完整工程代码可参考本书源码中的 chapter5/VSOpenCV4Demo1，建议读者先下载源码将工程运行起来，然后结合本书进行学习。

### 1. OpenCV 4 显示图像

首先加载 OpenCV 4 的头文件，代码如下：

```
#include <opencv2/opencv.hpp>
#include <iostream>
```

然后引入 OpenCV 4 和 C++的命名空间，代码如下：

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

Mat 类是 OpenCV 4 最基础的类，用来存储矩阵数据。创建一个 Mat 类型的变量 img，用它来存储图像，调用 cv::imread()函数读取一张图片，然后调用 cv::imshow()函数显示图片即可，其中 cv::imread()和 cv::imshow()的函数声明如下：

```
//chapter5/opencv4 - help - api.txt
/*
打开文件,可以指定图像的通道模式
@param filename Name of file to be loaded.
@param flags Flag that can take values of cv::ImreadModes
*/
CV_EXPORTS_W Mat imread(const String& filename, int flags = IMREAD_COLOR );

/*
显示图片,可以指定窗口名称,mat 表示需要显示的图片数据
@param winname Name of the window.
@param mat Image to be shown.
*/
CV_EXPORTS_W void imshow(const String& winname, InputArray mat);
```

该案例的完整代码如下：

```
//chapter5/VSOpenCV4Demo1/VSOpenCV4Demo1/VSOpenCV4Demo1.cpp
//VSOpenCV4Demo1.cpp : 定义控制台应用程序的入口点

#include "stdafx.h"
#include <opencv2/opencv.hpp>

int main(){
    //读取源图像并转换为灰度图像
    cv::Mat srcImage = cv::imread("fyxylogo.png");

    //判断文件是否读入正确
    if (!srcImage.data)
```

```
        return 1;

    //图像显示
    cv::imshow("srcImage", srcImage);
    //等待键盘键入
    cv::waitKey(0);
    return 0;
}
```

在该案例中，需要将 fyxylogo.png 图片放到源码路径下，如图 5-19 所示，然后单击“调试”菜单下的“开始执行(不调试)”，运行效果如图 5-20 所示。


**图 5-19　fyxylogo.png 图片的存放路径**


**图 5-20　OpenCV 4 显示 fyxylogo.png 图片**

## 2. 图像的极坐标转换

有时需要把图像或矩阵从直角坐标系(笛卡儿坐标系)转换到极坐标系，这个过程通常称为图像的极坐标变换。常见的作用是将一个圆形图像变换成一个矩形图像，类似于把圆剪开铺平。这样可以方便地处理钟表、圆盘等图像。图形上的圆形排列文字经过极坐标变换后可以垂直地排列在新图像上，便于对文字进行识别和检测，如图 5-21 所示。

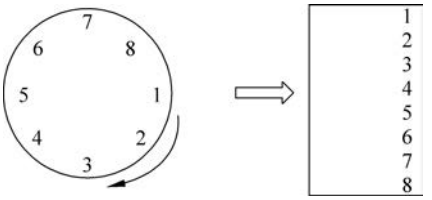


**图 5-21　图像的极坐标转换示意图**

OpenCV 4 中新增加了函数 warpPolar(),用于将图像或矩阵从直角坐标系(笛卡儿坐标系)转换到极坐标系,其 C++函数原型的代码如下:

```
//chapter5/opencv4 - help - api.txt
//图像的极坐标转换
void cv::warpPolar(InputArray src,
    OutputArray dst,
    Size dsize,
    Point2f center,
    double maxRadius,
    int flags
);
```

该函数的各个参数的含义如下。

(1) src:源图像,对通道数无要求,可以是灰度图像,也可以是彩色图像。

(2) dst:输出图像,它和源图像具有相同的数据类型和通道数。

(3) dsize:目标图像大小。

(4) center:极坐标变换时的原点坐标。

(5) maxRadius:极坐标系的极半径最大值。

(6) flags:插值方法与极坐标映射方法标志。两种方法之间通过“+”或者“|”号进行连接,其中插值方法通过一个枚举类型定义,代码如下:

```
//chapter5/opencv4 - help - api.txt
//! interpolation algorithm
enum InterpolationFlags{
    /** nearest neighbor interpolation */
    INTER_NEAREST = 0,
    /** bilinear interpolation */
    INTER_LINEAR = 1,
    /** bicubic interpolation */
    INTER_CUBIC = 2,
    //...省略代码
};
```

因为变换本质上是在离散序列中进行的,而不是连续的,这就导致两个坐标系的点与点之间并不能一一对应。为了尽可能地保证目标极坐标矩阵中的每个点的值的准确性,所以需要进行插值处理。下面通过一个案例来演示直角坐标系到极坐标系的转换,先显示一张

图像，然后通过 warpPolar()函数转换到极坐标，然后通过 warpPolar()函数又转换回直角坐标，代码如下：

```
//chapter5/opencv4 - help - api.txt
//polar 坐标转换
int main_polar_demo(){
    Mat img = imread("clock_dial.jpg");
    Mat img1, img2;
    Point2f center = Point2f(img.cols / 2, img.rows / 2);

    //将直角坐标系图像转换为极坐标系图像
    warpPolar(img, img1, Size(300, 600), center, center.x,
        INTER_LINEAR + WARP_POLAR_LINEAR);

    //将极坐标系图像转换为直角坐标系图像
    warpPolar(img1, img2, Size(img.rows, img.cols), center, center.x,
        INTER_LINEAR + WARP_POLAR_LINEAR + WARP_INVERSE_MAP);

    imshow("原图", img);
    imshow("直角坐标→极坐标", img1);
    imshow("极坐标→直角坐标", img2);
    waitKey(0);
    return 0;
}
```

编译并运行该程序，效果如图 5-22 所示，从左到右分别是原图、极坐标图及直角坐标图。

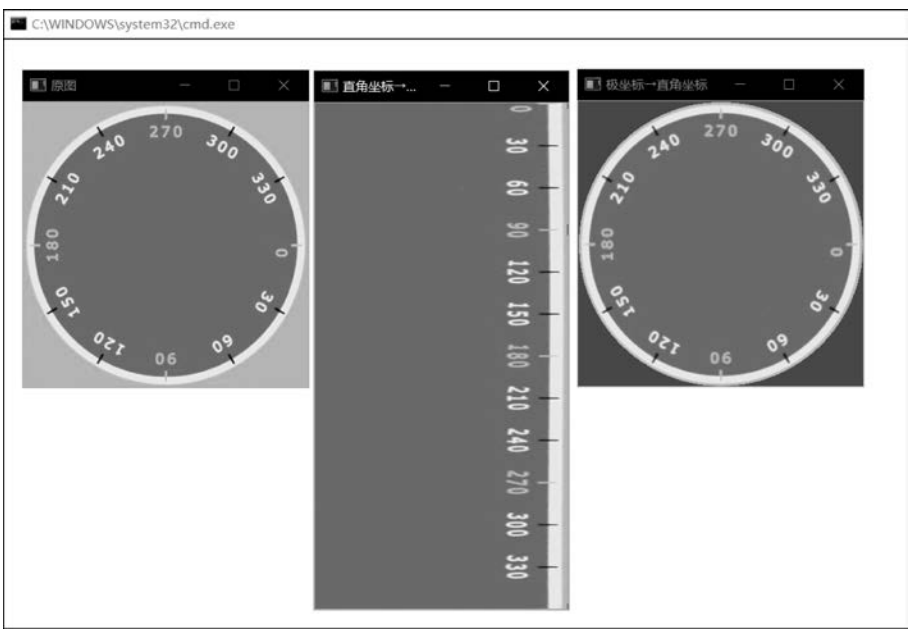


**图 5-22　坐标转换效果**

### 3. OpenCV 4 采集并显示摄像头

使用 OpenCV 4 的 VideoCapture 类可以访问本地摄像头，先打开摄像头，然后一张一张地读取视频帧，显示到窗口上即可，主要代码如下：

```
//chapter5/VSOpenCV4Demo1/VSOpenCV4Demo1/VSOpenCV4Demo1.cpp
//采集并显示摄像头
int main_camera_show(){
    //用 Videocapture 结构创建一个 camera 视频对象
    VideoCapture camera;

    //打开视频摄像头
    camera.open(0);                     //该方式会报错
    //camera.open(0, cv::CAP_DSHOW);    //以 DShow 方式打开，成功
    if (!camera.isOpened()) {
        printf("could not load video data...\n");
        return -1;
    }

    //获取视频帧数目(一帧就是一张图片):摄像头会返回 -1
    int frames = camera.get(CAP_PROP_FRAME_COUNT);
    //获取每帧视频的频率
    double fps = camera.get(CAP_PROP_FPS);
    //获取帧的视频宽度和视频高度
    Size size = Size(camera.get(CAP_PROP_FRAME_WIDTH),
        camera.get(CAP_PROP_FRAME_HEIGHT));
    cout << frames << endl;
    cout << fps << endl;
    cout << size << endl;

    //创建视频中每张图片对象
    Mat frame;

    //循环显示视频中的每张图片
    for (;;){
        //将视频转给每幅图进行处理
        camera >> frame;

        //视频播放完退出
        if (frame.empty())break;
        imshow("video-demo", frame);

        //在视频播放期间按键退出
        if (waitKey(33) >= 0) break;
    }
    //释放
    camera.release();
    return 0;
}
```

分析代码可知，先定义 VideoCapture 类型的变量 camera，通过 open()函数打开摄像

头，也可以通过 get()函数获取摄像头的参数(如帧率、宽和高等)。通过 VideoCapture 的重载操作符>>来读取一帧视频图像，然后调用 imshow()函数显示到窗口上。编译并运行该程序，会发现运行时报错，如图 5-23 所示。

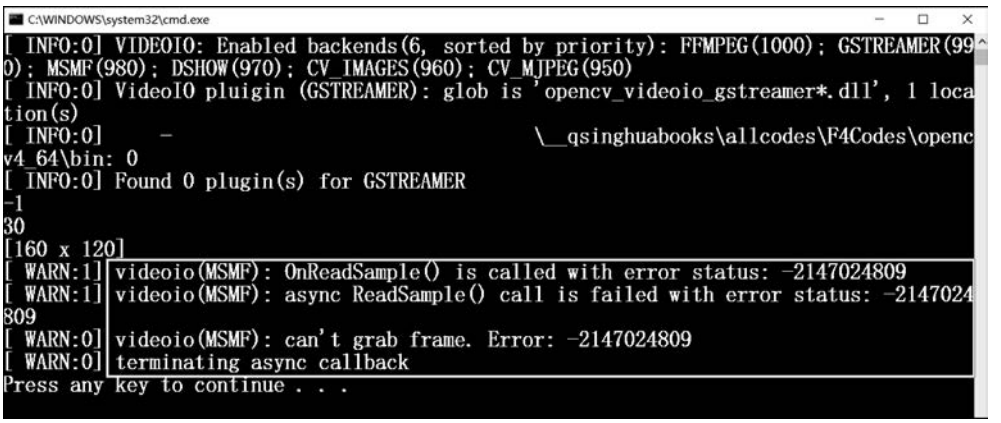


**图 5-23　OpenCV 打开摄像头失败**

videoio(MSMF)：can't grab frame. Error：－2147024809，这个报错信息表示抓帧失败。其实这个问题与 USB 相机的 ID 号有关，代码中使用默认的 0 来打开摄像头，代码如下：

```
camera.open(0);
```

OpenCV 4 默认的 VideoCapture 在打开摄像头时会使用默认方式，即直接填写 ID，但是，如果项目与 dshow 相关，则 dshow 显示的摄像头顺序与 OpenCV 4 默认的打开顺序不同，所以需要使用第 2 个参数与之对应，代码如下：

```
camera.open(0, cv::CAP_DSHOW);
```

下面看一下 VideoCapture 类的 open()函数，其函数原型如下：

```
//chapter5/opencv4 - help - api.txt
/** 打开视频摄像头
    参数与构造函数 VideoCapture 相同
    如果相机已成功打开，则返回 true
    该方法首先调用 VideoCapture::release()函数来关闭已打开的文件或相机
    */
    CV_WRAP virtual bool open(int index, int apipreference = CAP_ANY);
```

其中第 2 个参数是枚举类型，代码如下：

```
//chapter5/opencv4 - help - api.txt
enum VideoCaptureAPIs {
```

```
    CAP_ANY          = 0,        //!< 自动检测
    CAP_VFW          = 200,      //!< Windows 系统中的 VFM 模式(obsolete, removed)
    CAP_V4L          = 200,      //!< Linux 系统中的 V4L/V4L2 音视频捕获
    CAP_V4L2         = CAP_V4L,  //!< Linux 系统中的 V4L/V4L2 音视频捕获
    CAP_QT           = 500,      //!< macOS 系统中的音视频捕获(obsolete, removed)
    CAP_DSHOW        = 700,      //!< Windows 系统中 DirectShow 音视频捕获(via videoInput)
    //.....省略代码
};
```

所以归根结底在打开摄像头时需要进一步指定 DShow(DirectShow)来打开,其对应的 ID 值是 700,代码如下:

```
//camera.open(0);                //该方式会报错
camera.open(0, cv::CAP_DSHOW);   //以 DShow 方式打开,成功
```

重新编译并运行程序,会在控制台输出相关信息,例如帧数为-1、帧率为 30、分辨率为 640×480,并弹出一个新窗口显示摄像头捕获的视频帧,如图 5-24 所示。


**图 5-24　使用 OpenCV 4 读取并显示摄像头**

通过 VideoCapture 的 get 函数可以获取摄像头的各类参数,传入的参数是枚举类型,代码如下:

```
//chapter5/opencv4 - help - api.txt
enum VideoCaptureProperties {
      CAP_PROP_POS_MSEC = 0,          //!< 视频文件的当前位置(单位为毫秒)
      CAP_PROP_POS_FRAMES = 1,        //!< 接下来要解码/捕获的帧的基于 0 的索引
      CAP_PROP_POS_AVI_RATIO = 2,     //!< 视频文件的相对位置: 0 = 影片开始,1 = 影片结束
      CAP_PROP_FRAME_WIDTH = 3,       //!< 宽度
      CAP_PROP_FRAME_HEIGHT = 4,      //!< 高度
      CAP_PROP_FPS = 5,               //!< 帧率
      CAP_PROP_FOURCC = 6,            //!< 编解码器的 4 个字符代码,请查阅
      CAP_PROP_FRAME_COUNT = 7,       //!< 视频文件中的帧数
```

```
    //...
};
```

### 4. OpenCV 4 实现轮廓效果

图像的边缘由图像中两个相邻的区域之间的像素集合组成，是指图像中一个区域的结束和另外一个区域的开始。也可以理解为，图像边缘就是图像中灰度值发生空间突变的像素的集合。梯度方向和幅度是图像边缘的两个性质，沿着跟边缘垂直的方向，像素值的变化幅度比较平缓，而沿着与边缘平行的方向，则像素值变化幅度比较大。于是，根据该变化特性，通常会采用计算一阶或者二阶导数的方法来描述和检测图像边缘。

基于边缘检测的图像分割方法的基本思路是首先检测出图像中的边缘像素，然后把这些边缘像素集合连接在一起便组成所要的目标区域边界。图像中的边缘可以通过对灰度值求导来检测确定，然而求导数可以通过计算微分算子实现。在数字图像处理领域，微分运算通常被差分计算所近似代替。使用 OpenCV 的 Canny()函数可以检测到图像的轮廓，并进行二值化处理，函数原型如下：

```
//chapter5/opencv4 - help - api.txt
CV_EXPORTS_W void Canny(InputArray image, OutputArray edges,
                    double threshold1, double threshold2,
                    int apertureSize = 3, bool L2gradient = false );
```

该函数的各个参数的含义如下。

(1) image：8 位输入图像。

(2) edges：输出边缘，单通道 8 位图像，与图像大小相同。

(3) threshold1：迟滞过程的第 1 个阈值。

(4) threshold2：迟滞过程的第 2 个阈值。

(5) apertureSize：Sobel 算子的孔径大小。

(6) L2gradient：一个标志值，指示是否应用更精确的方式计算图像梯度幅值。

使用 OpenCV 4 读取摄像头的视频帧之后，调用 Canny()函数即可完成轮廓的提取，代码如下：

```
//chapter5/VSOpenCV4Demo1/VSOpenCV4Demo1/VSOpenCV4Demo1.cpp
//摄像头 + Canny
int main_camera_Canny(){
    //用 VideoCapture 结构创建一个 capture 视频对象
    VideoCapture capture;
    //连接视频摄像头
    //capture.open(0);                                    //error
    capture.open(0, CAP_DSHOW);
    if (!capture.isOpened()) {
        printf("could not load video data...\n");
        return -1;
```

```
        }
        int frames = capture.get(CAP_PROP_FRAME_COUNT);          //获取视频帧数目
        double fps = capture.get(CAP_PROP_FPS);                  //获取每帧视频的频率
        //获取帧的视频宽度和视频高度
        Size size = Size(capture.get(CAP_PROP_FRAME_WIDTH),
            capture.get(CAP_PROP_FRAME_HEIGHT));
        cout << frames << endl;
        cout << fps << endl;
        cout << size << endl;
        //创建视频中每张图片对象
        Mat frame;
        namedWindow("video - src", WINDOW_AUTOSIZE);
        //循环显示视频中的每张图片
        Mat edgeMat;

        for (;;){
            //将视频转给每张图进行处理
            capture >> frame;

            //视频播放完退出
            if (frame.empty())break;
            imshow("video - src", frame);

            //检测边缘图像,并二值化
            Canny(frame, edgeMat, 80, 180, 3, false);
            imshow("edge", edgeMat);

            //在视频播放期间按键退出
            if (waitKey(33) >= 0) break;
        }
        //释放
        capture.release();
        return 0;
    }
```

编译并运行上述代码,可以提取图像中的轮廓并进行二值化(黑白图)处理,如图 5-25 所示。

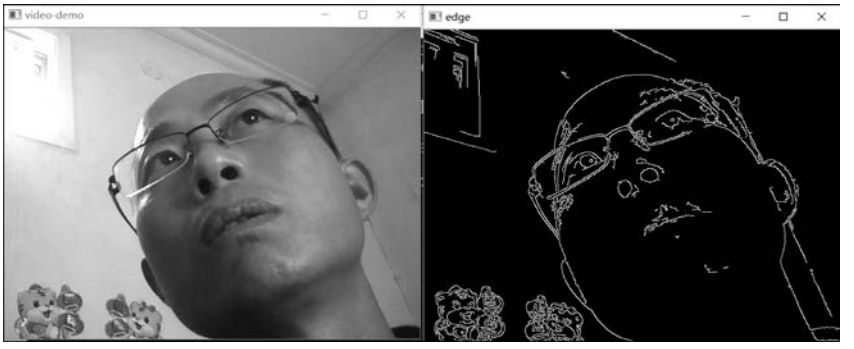


**图 5-25　OpenCV 4 提取摄像头数据的轮廓**

### 5. OpenCV 4 实现腐蚀效果

形态学(Morphology)一词通常表示生物学的一个分支，该分支主要研究动植物的形态和结构，而图像处理中所指的形态学，往往表示的是数学形态学(Mathematical Morphology)。它是一门建立在格论和拓扑学基础之上的图像分析学科，是数学形态学图像处理的基本理论。其基本的运算包括二值腐蚀和膨胀、二值开闭运算、骨架抽取、极限腐蚀、击中击不中变换、形态学梯度、顶帽(Top-hat)变换、颗粒分析、流域变换、灰值腐蚀和膨胀、灰值开闭运算、灰值形态学梯度等。

1) 形态学中膨胀与腐蚀

简单来讲，形态学操作就是基于形状的一系列图像处理操作。OpenCV 为进行图像的形态学变换提供了快捷方便的函数，最基本的形态学操作有两种：膨胀(Dilation)和腐蚀(Erosion)，使用 dilate()和 erode()函数即可完成这两种操作。膨胀就是求局部最大值的操作。按数学知识来讲，膨胀或者腐蚀操作就是将图像(或图像的一部分区域，称为 A)与核(称为 B)进行卷积。核可以是任何的形状和大小，它拥有一个单独定义出来的参考点，称其为锚点(Anchor Point)。多数情况下，核是一个小的中间带有参考点的实心正方形或者圆盘，其实，可以把核视为模板或者掩码，而膨胀就是求局部最大值操作，核 B 与图形卷积，即计算核 B 覆盖的区域的像素的最大值，并把这个最大值赋值给参考点指定的像素。这样就会使图像中的高亮区域逐渐增长。膨胀和腐蚀是相反的一对操作，所以腐蚀就是求局部最小值操作，一般会把腐蚀和膨胀对应起来理解和学习。

2) 图像卷积与卷积核

图像卷积操作可以看成一个窗口区域在另外一个大的图像上移动，对每个窗口覆盖的区域都进行点乘得到的值作为中心像素的输出值。窗口的移动是从左到右，从上到下的。窗口可以理解成一个指定大小的二维矩阵，里面有预先指定的值，该过程如图 5-26 所示。

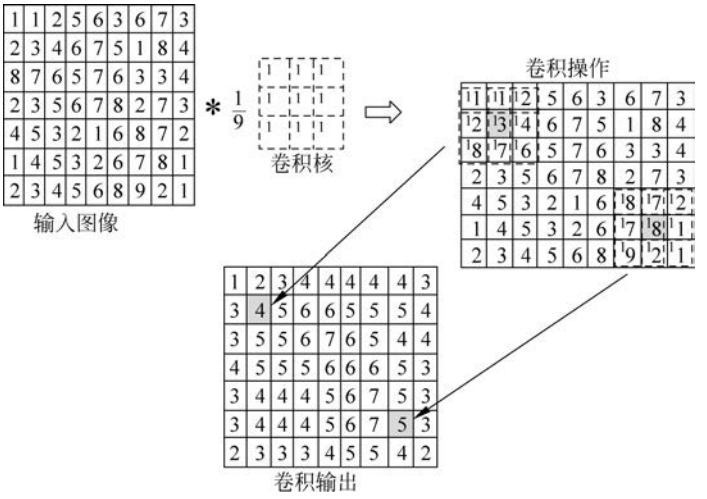


**图 5-26　图像卷积操作**

图像滤波是在尽量保留图像细节特征的条件下对目标图像的噪声进行抑制，是图像预处理中不可缺少的操作，其处理效果的好坏将直接影响后续图像处理和分析的有效性和可靠性。线性滤波是图像处理最基本的方法，它允许对图像进行处理，产生很多不同的效果。首先，需要一个二维的滤波器矩阵(卷积核)和一个要处理的二维图像，然后对于图像的每个像素，计算它的邻域像素和滤波器矩阵的对应元素的乘积，最后加起来，作为该像素位置的值。这样就完成了滤波过程。对图像和滤波矩阵进行逐个元素相乘再求和的操作就相当于将一个二维的函数移动到另一个二维函数的位置，这个操作就叫卷积，其中卷积核的定义规则如下：

(1) 滤波器的大小应该是奇数，这样它才有一个中心，例如 3×3、5×5 或者 7×7。有中心了，也就有了半径的称呼，例如 5×5 大小的核对应的半径就是 2。

(2) 滤波器矩阵所有的元素之和应该等于 1，这是为了保证滤波前后图像的亮度保持不变。注意这不是硬性要求。

(3) 如果滤波器矩阵所有元素之和大于 1，则滤波后的图像就会比原图像更亮，反之，如果小于 1，则得到的图像就会变暗。如果和为 0，则图像不会变黑，但也会非常暗。

(4) 对于滤波后的结构，可能会出现负数或者大于 255 的数值。对这种情况，将它们直接截断到 0～255 即可。对于负数，也可以取绝对值。

在 OpenCV 甚至平常的图像处理中，卷积核是一种最常用的图像处理工具。其主要通过确定的核块来检测图像的某个区域，之后根据所检测的像素与其周围存在的像素的亮度差值来改变像素明亮度，例如一个卷积核的伪代码如下：

```
Kernel33 = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
```

这是一个[3,3]的卷积核，其作用是计算中央像素与周围邻近像素的亮度差值，如果亮度差值的差距过大，本身图像的中央亮度较低，则经过卷积核以后，中央像素的亮度会增加，即如果一像素比周围的像素更加突出，则提升其本身的亮度。

3) OpenCV 4 实现腐蚀操作

erode()函数使用像素邻域内的局部极小运算符来腐蚀图像，函数原型如下：

```
//chapter5/opencv4-help-api.txt
void erode(InputArray src, OutputArray dst, InputArray Kernel, Point anchor = Point(-1, -1),
int iterations = 1, int borderType = BORDER_CONSTANT, const Scalar& borderValue =
morphologyDefaultBorderValue());
```

该函数的各个参数的含义如下。

(1) InputArray 类型的 src：输入图像，Mat 类的对象即可。图像的通道数可以是任意的，但是图像的深度应该是 CV_8U、CV_16U、CV_16S、CV_32F 或 CV_64F 中的一个。

(2) OutputArray 类型的 dst：目标图像，需要和输入图像有相同的尺寸和类型。

(3) InputArray 类型的 Kernel：膨胀操作的核。当为 NULL 时，表示使用的是参考点位于中心 3×3 的核。

(4) Point 类型的 anchor：锚点的位置，默认值为 (－1，－1)，表示位于中心。

(5) int 类型的 iterations：迭代的次数，默认值为 1。

(6) int 类型的 borderType：用于推断图像外部像素的某种边界模式，默认值为 BORDER_DEFAULT。

(7) const Scalar& 类型的 borderValue：一般不管它。

一般只需传入前 3 个参数，后面的 4 个参数有默认值。使用 erode()函数的案例代码如下：

```
//chapter5/VSOpenCV4Demo1/VSOpenCV4Demo1/VSOpenCV4Demo1.cpp
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace std;
using namespace cv;
//腐蚀效果
int main_erode_demo() {
    Mat srcImage;
    srcImage = imread("./fyxylogo.png");                    //读取图片
    imshow("picture-src", srcImage);

    Mat element;
    element = getStructuringElement(MORPH_RECT, Size(5, 5));//卷积核

    Mat dstImage;
    erode(srcImage, dstImage, element);                     //腐蚀操作
    imwrite("erode.jpg", dstImage);                         //将腐蚀后的 Mat 写入本地文件
    imshow("picture-rode", dstImage);

    waitKey(0);
    return 0;
}
```

在该案例中，使用了 getStructuringElement()函数，它会返回指定形状和尺寸的结构元素，函数原型如下：

```
Mat getStructuringElement(int shape, Size esize, Point anchor = Point(-1, -1));
```

该函数的第 1 个参数表示内核的形状，有 3 种形状可以选择，如下所示。

(1) 矩形：MORPH_RECT。

(2) 交叉形：MORPH_CROSS。

(3) 椭圆形：MORPH_ELLIPSE。

第 2 个和第 3 个参数分别是内核的尺寸及锚点的位置。一般在调用 erode()及 dilate()函数之前，先定义一个 Mat 类型的变量来获得 getStructuringElement()函数的返回值。对

于锚点的位置，有默认值 Point(－1，－1)，表示锚点位于中心点。element 形状唯一依赖锚点位置，其他情况下，锚点只是影响了形态学运算结果的偏移。编译并运行该程序，效果如图 5-27 所示，同时会通过 imwrite()函数生成一张图片(erode.jpg)。

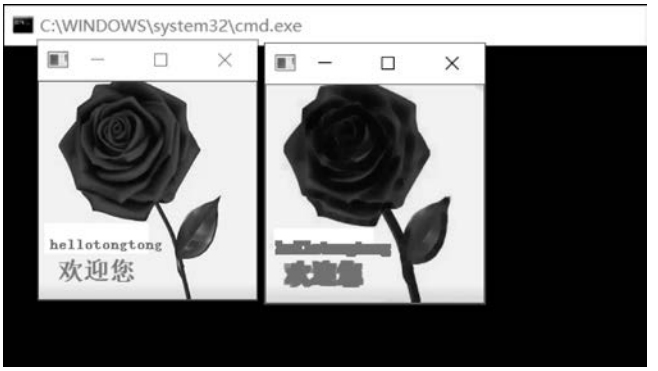


**图 5-27　OpenCV 4 实现图片的腐蚀操作**

4) OpenCV 4 对摄像头捕获的视频进行腐蚀操作

使用 OpenCV 4 读取摄像头的视频帧之后，可以调用 erode()函数进行腐蚀操作，代码如下：

```
//chapter5/VSOpenCV4Demo1/VSOpenCV4Demo1/VSOpenCV4Demo1.cpp
//摄像头 + 膨胀、腐蚀
int main_camera_erode(){
    //用 VideoCapture 结构创建一个 capture 视频对象
    VideoCapture capture;
    //打开摄像头
    capture.open(0);
    if (!capture.isOpened()) {
        printf("could not load video data...\n");
        return -1;
    }
    //获取帧的视频宽度、视频高度、分辨率、帧率等
    int frames = capture.get(CAP_PROP_FRAME_COUNT);              //帧数:摄像头会返回-1
    double fps = capture.get(CAP_PROP_FPS);
    Size size = Size(capture.get(CAP_PROP_FRAME_WIDTH),
                     capture.get(CAP_PROP_FRAME_HEIGHT));
    cout << frames << endl;
    cout << fps << endl;
    cout << size << endl;
    //创建视频中每张图片对象
    Mat frame;
    //namedWindow("video-demo", WINDOW_AUTOSIZE);
    //循环显示视频中的每张图片
    Mat edge;

//获取结构元素，定义卷积核
Mat element = getStructuringElement(MORPH_RECT, Size(7, 7), Point(-1, -1));
```

```
    Mat image_out; //膨胀或腐蚀后的 Mat

    for (;;){
        //将视频转给每张图进行处理
        capture >> frame;
        //省略对图片的处理
        //视频播放完退出
        if (frame.empty())break;
        imshow("video - src",frame);

        //dilate(frame, image_out, element, Point( - 1,  - 1), 1);          //膨胀
        erode(frame, image_out, element, Point( - 1,  - 1), 1);             //腐蚀
        imshow("image_out", image_out);

        //在视频播放期间按键退出
        if (waitKey(1000/fps) >=  0) break;
    }
    //释放
    capture.release();
    return 0;
}
```

编译并运行上述代码，会对摄像头捕获的视频帧进行腐蚀操作，如图 5-28 所示。

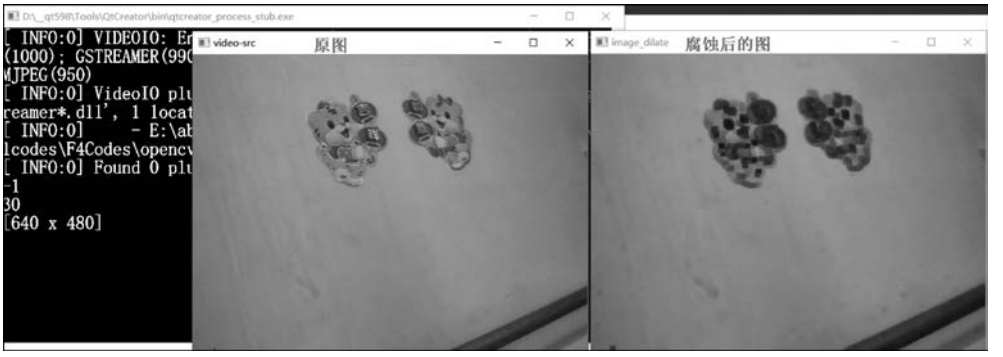


**图 5-28　OpenCV 4 的摄像头腐蚀操作**

### 6. OpenCV 4 实现磨皮美颜效果

皮肤美化处理主要包括磨皮和美白，磨皮需要把脸部皮肤区域处理得细腻、光滑，美白则需要将皮肤区域处理得白皙、红润。磨皮主要通过保边滤波器对脸部非器官区域进行平滑，达到脸部皮肤区域光滑的效果。一般来讲常用的保边滤波器主要有双边滤波、导向滤波、表面模糊滤波、局部均值滤波等方法，考虑到性能和效果的平衡，一般采用双边滤波(Bilateral Filter)或者导向滤波。双边滤波考虑了窗口区域内像素的欧氏距离和像素强度差异这两个维度，使其在进行平滑时具有保护边缘的特性。其优点是在 GPU 侧计算量小、资源消耗低，其缺点是无法去除色差较大的孤立点，如痘痘、黑痣等，并且磨皮后的效果较为生硬。而导向滤波则会根据窗口区域内纹理的复杂程度进行平滑程度的调节，在平坦区域

趋近于均值滤波，在纹理复杂的区域则趋近于原图，窗口区域内纹理的复杂程度跟均值和方差强相关，既能够很好地处理平坦区域的各种噪点，又能较完整地保存好轮廓区域的信息，并且在 GPU 侧的计算并不复杂。

双边滤波是一种非线性的滤波方法，是结合图像的空间邻近度和像素值相似度的一种折中处理，同时考虑空域信息和灰度相似性，达到保边去噪的目的。具有简单、非迭代、局部的特点。双边滤波的好处是可以做边缘保存(Edge Preserving)，一般用高斯滤波去降噪，会较明显地模糊边缘，对于高频细节的保护效果并不明显。双边滤波顾名思义比高斯滤波多了一个高斯方差，它是基于空间分布的高斯滤波函数，所以在边缘附近，离得较远的像素不会过多地影响到边缘上的像素值，这样就保证了边缘附近像素值的保存，但是由于保存了过多的高频信息，对于彩色图像里的高频噪声，双边滤波不能干净地滤掉，只能对低频信息进行较好滤波。OpenCV 4 提供了双边滤波的函数，原型如下：

```
//chapter5/opencv4 - help - api.txt
void bilateralFilter(InputArray src,
                OutputArray dst,
                int d,
                double sigmaColor,
                double sigmaSpace,
                int borderType = BORDER_DEFAULT );
```

该函数的各个参数的含义如下。

(1) InputArray 类型的 src：输入图像，即源图像，需要为 8 位或者浮点型单通道、三通道的图像。

(2) OutputArray 类型的 dst：即目标图像，需要和源图片有相同的尺寸和类型。

(3) int 类型的 d：表示在过滤过程中每个像素邻域的直径。如果将这个值设为非正数，则 OpenCV 会从第 5 个参数 sigmaSpace 来把它计算出来。

(4) double 类型的 sigmaColor：颜色空间滤波器的 sigma 值。这个参数的值越大，就表明该像素邻域内有更宽广的颜色会被混合到一起，产生较大的半相等颜色区域。

(5) double 类型的 sigmaSpace：坐标空间中滤波器的 sigma 值，坐标空间的标准方差。它的数值越大，意味着越远的像素会相互影响，从而使更大的区域足够相似的颜色可获取相同的颜色。当 d>0，d 指定了邻域大小且与 sigmaSpace 无关。否则 d 正比于 sigmaSpace。

(6) int 类型的 borderType：用于推断图像外部像素的某种边界模式。注意它有默认值 BORDER_DEFAULT。

使用 OpenCV 读取摄像头的视频帧之后，调用 bilateralFilter()函数即可完成双边滤波的效果，代码如下：

```
//chapter5/VSOpenCV4Demo1/VSOpenCV4Demo1/VSOpenCV4Demo1.cpp
//摄像头 + 磨皮美颜
int main_camera_beauty(){
    //用 Videocapture 结构创建一个 capture 视频对象
```

```
    VideoCapture capture;
    //连接视频
    capture.open(0, cv::CAP_DSHOW);
    if (!capture.isOpened()) {
        printf("could not load video data...\n");
        return -1;
    }
    int frames = capture.get(CAP_PROP_FRAME_COUNT);      //获取视频帧数目
    double fps = capture.get(CAP_PROP_FPS);              //获取每帧视频的频率
    //获取帧的视频宽度和视频高度
    Size size = Size(capture.get(CAP_PROP_FRAME_WIDTH),
        capture.get(CAP_PROP_FRAME_HEIGHT));
    cout << frames << endl;
    cout << fps << endl;
    cout << size << endl;
    //创建视频中每张图片对象
    Mat frame;
    namedWindow("video-demo", WINDOW_AUTOSIZE);
    //循环显示视频中的每张图片
    for (;;){
        //将视频转给每张图进行处理
        capture >> frame;
        //...
        //视频播放完退出
        if (frame.empty())break;
        imshow("video-src", frame);

        Mat bila_image;
        bilateralFilter(frame, bila_image, 0, 100, 10, 4);
        //pyrMeanShiftFiltering(frame, bila_image, 15, 30);
        imshow("bila_image", bila_image);
        //在视频播放期间按键退出
        if (waitKey(33) >= 0) break;
    }
    //释放
    capture.release();
    return 0;
}
```

编译并运行上述代码，可以实现磨皮美颜的效果，如图 5-29 所示。

**注意**：使用 OpenCV 4 的 bilateralFilter()函数可以进行磨皮美颜，但效果一般，更好的磨皮美颜算法需要研究专业的论文。同时，在直播场景下还要考虑性能问题。

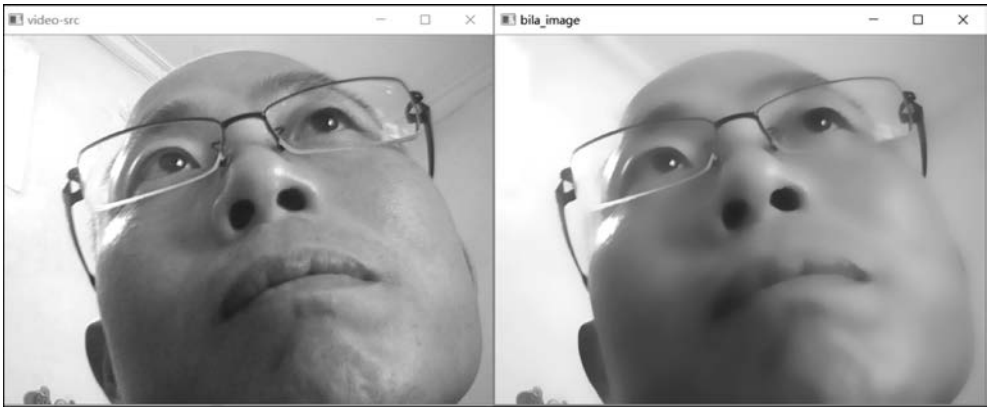


图 5-29　OpenCV 4 的双边滤波及磨皮效果

## 5.3　MFC 结合 OpenCV 显示图片

5.2 节通过 Win32 控制台应用程序来调用 OpenCV 4 的 API，但是窗口都是独立弹出来的。这种方式做测试还可以，真实项目中一般采用 GUI 程序，例如使用 MFC 框架开发的窗口程序，所以将 OpenCV 4 创建的窗口嵌入 MFC 的控件上。

**注意**：本节案例的完整工程代码可参考本书源码中的 chapter5/MFCOpenCV 4Demo2，建议读者先下载源码将工程运行起来，然后结合本书进行学习。

### 1. MFC＋OpenCV 4 显示图像

由于 OpenCV 4 常用的界面只是单纯地打开图像窗口，相关界面控件和工具较少且不美观，故使用 MFC 制作界面，而用 OpenCV 4 做图像处理。此时便需要在 MFC 中显示 OpenCV 4 所用的图片。使用 VS 2015 新建 MFC 类型的应用程序，在项目名称后输入 MFCOpenCV 4Demo2，单击“确定”按钮，如图 5-30 所示。在弹出的“欢迎使用 MFC 应用程序向导”页面，单击“下一步”按钮，如图 5-31 所示，然后在“应用程序类型”页面选择“基于对话框”的选项类型，单击“完成”按钮，如图 5-32 所示。项目创建之后，切换到 x64 开发模式，然后配置 OpenCV 4 的包含目录、库目录和附加依赖项，这里不再赘述，如图 5-33 所示。

双击 MFCOpenCV 4Demo2. rc 文件进入“资源视图”选项卡，然后双击 IDD_MFCOPENCV4DEMO2_DIALOG 打开对话框的界面设计器，从左侧“工具箱”中将一个 Picture Control 控件拖曳到对话框界面上，然后右击该控件，在弹出的菜单中单击“属性”，在右侧的“属性”列表中将 ID 修改为 IDC_STATIC_pic1，如图 5-34 所示。将 OpenCV 4 的窗口嵌入 MFC 的 Picture Control 控件上，只需将 OpenCV 窗口的父窗口句柄设置为 Picture Control 控件的句柄，核心代码如下：

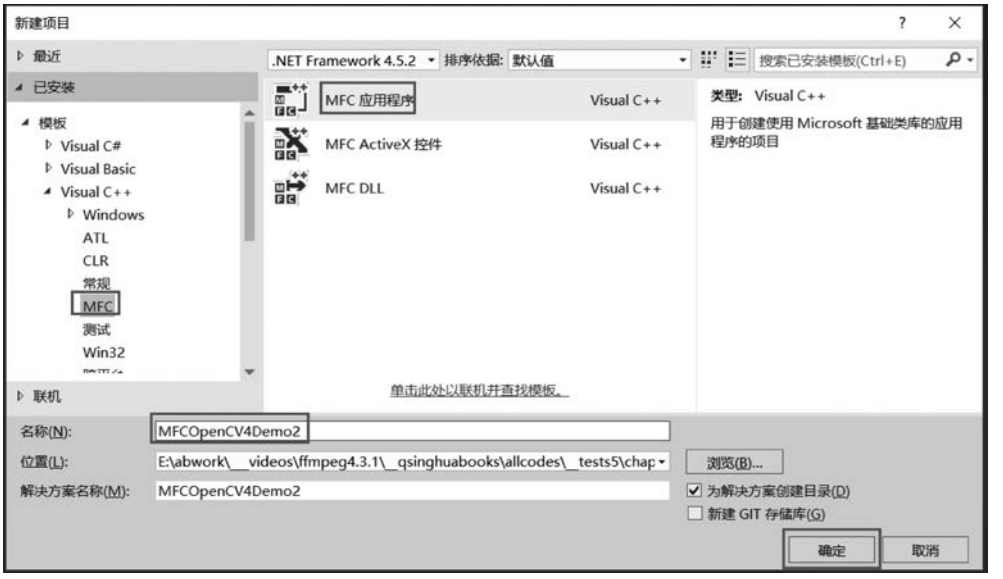


图 5-30　创建 MFC 应用程序


图 5-31　单击“下一步”按钮


图 5-32　选择“基于对话框”

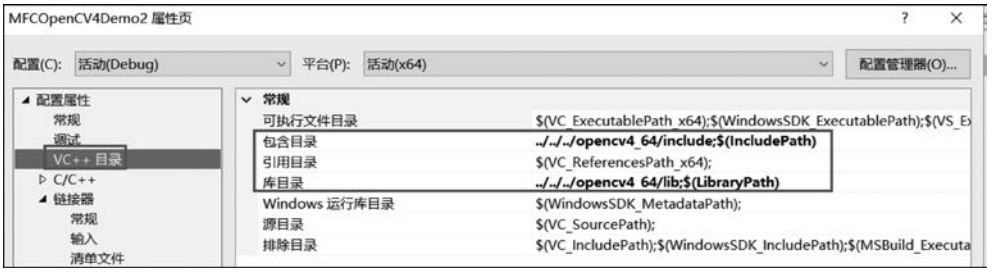


图 5-33　配置项目的包含目录和库目录

```
//chapter5/opencv4 - help - api.txt
# define OPENCV_WINDOW_NAME_PIC1 "OCImageShow1"
namedWindow(OPENCV_WINDOW_NAME_PIC1);                    //创建 OpenCV 窗口

//hWndOpenCV 表示窗口句柄,获取窗口句柄(若显示 cvGetWindowHandle 未定义
//则需要添加 # include opencv2/highgui/highgui_c.h 头文件)
//获取 OpenCV 窗口的句柄
HWND hWndOpenCV = (HWND)cvGetWindowHandle(OPENCV_WINDOW_NAME_PIC1);

//GetParent 函数用于获取一个指定子窗口的父窗口句柄
HWND hParent = ::GetParent(hWndOpenCV);
```

```
::ShowWindow(hParent, SW_HIDE);              //隐藏这个默认的父窗口

//CWnd 是 MFC 窗口类的基类,提供了微软基础类库中所有窗口类的基本功能
//将 OpenCV 窗口的父窗口句柄设置为 Picture Control 控件
::SetParent(hWndOpenCV, GetDlgItem(IDC_STATIC_pic1)->m_hWnd);
```

分析上述代码发现,首先调用 namedWindow()函数创建一个 OpenCV 的窗口,然后调用 cvGetWindowHandle()函数获取该窗口的句柄。获得该窗口句柄后,再调用 GetParent()函数获取它的父窗口句柄,然后调用 ShowWindow()函数将这个父窗口隐藏。最后调用 SetParent()函数给刚才创建的 OpenCV 窗口设置新的父窗口,即设置为 MFC 的 Picture Control 控件。

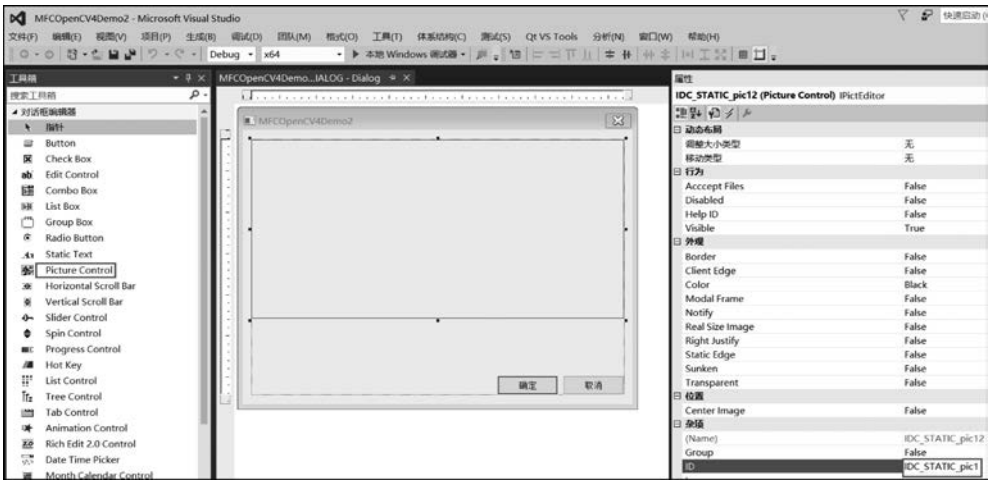


**图 5-34　往对话框界面上拖曳一个 Picture Control 控件**

在 MFCOpenCV 4Demo2Dlg.cpp 文件的开头处需要加入 OpenCV 的头文件,这里需要特别注意 highgui_c.h 这个头文件,因为 cvGetWindowHandle()函数会用到 highgui_c.h 头文件,代码如下:

```
//chapter5\MFCOpenCV4Demo2\MFCOpenCV4Demo2/MFCOpenCV4Demo2Dlg.h
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui_c.h>
using namespace cv; //包含 cv 命名空间
using namespace std;
```

然后在 CMFCOpenCV 4Demo2Dlg::OnInitDialog()函数中添加的代码如下:

```
//chapter5\MFCOpenCV4Demo2\MFCOpenCV4Demo2/MFCOpenCV4Demo2Dlg.cpp
//TODO: 在此添加额外的初始化代码
//定义宏,OpenCV 的窗口名称
#define OPENCV_WINDOW_NAME_PIC1 "OCImageShow1"
```

```
namedWindow(OPENCV_WINDOW_NAME_PIC1);                    //创建 OpenCV 窗口

//hWndOpenCV 表示窗口句柄,获取窗口句柄(若显示 cvGetWindowHandle 未定义
//则需要添加 # include opencv2/highgui/highgui_c.h 头文件)
//先获取 OpenCV 窗口的句柄
HWND hWndOpenCV = (HWND)cvGetWindowHandle(OPENCV_WINDOW_NAME_PIC1);
//GetParent 函数用于获取一个指定子窗口的父窗口句柄
HWND hParent = ::GetParent(hWndOpenCV);

//CWnd 是 MFC 窗口类的基类,提供了微软基础类库中所有窗口类的基本功能
//将 OpenCV 窗口的父窗口设置为 Picture Control 控件
 //这句,将 OpenCV 窗口嵌入了 Picture Control 控件中
::SetParent(hWndOpenCV, GetDlgItem(IDC_STATIC_pic1)->m_hWnd);

//ShowWindow 指定窗口中显示或隐藏,这里隐藏 OpenCV 窗口的默认父窗口
::ShowWindow(hParent, SW_HIDE);

//通过 imread 函数读取本地图片
Mat srcImg = imread("fyxylogo.jpg");                     //OpenCV 读取图片

//调用 imshow 将 Mat 图片数据显示到指定窗口
imshow(OPENCV_WINDOW_NAME_PIC1, srcImg);                 //OpenCV 显示图片
```

编译并运行该程序,效果如图 5-35 所示。


**图 5-35　将 OpenCV 窗口嵌入 MFC 控件上**

## 2. MFC＋OpenCV 4 二值化处理图像

图像二值化(Image Binarization)就是将图像上的像素的灰度值设置为 0 或 255,也就是将整个图像呈现出明显的黑白效果的过程。在数字图像处理中,二值图像占有非常重要的地位,图像的二值化使图像的数据量减小,从而凸显出目标的轮廓。它的原理是将 256 个亮度等级的灰度图像通过适当的阈值选取而获得仍然可以反映图像整体和局部特征的二值化图像。首先,图像的二值化有利于图像的进一步处理,使图像变得简单,而且数据量减小,

能凸显出感兴趣的目标的轮廓。其次，要进行二值图像的处理与分析，一般需要把灰度图像二值化，得到二值化图像。为了得到理想的二值图像，一般采用封闭、连通的边界定义不交叠的区域。所有灰度大于或等于阈值的像素被判定为属于特定物体，其灰度值用 255 表示，否则这些像素会被排除在物体区域以外，灰度值为 0，表示背景或者例外的物体区域。OpenCV 4 提供了 threshold()函数，用于二值化处理，函数原型如下：

```
//chapter5/opencv4 - help - api.txt
/* 二值化处理函数
@param src: 输入阵列(多通道、8 位或 32 位浮点)
@param dst: 与 src 具有相同大小和类型及相同通道数的输出数组
@param thresh: 阈值
@param maxval: 与 # THRESH_BINARY 和 # THRESH_BINARY_INV 阈值类型一起使用的最大值
@param type: 阈值类型 (请参见 # ThresholdTypes).
@如果使用 Otsu 或 Triangle 方法，则返回计算的值

@sa adaptiveThreshold, findContours, compare, min, max
*/
CV_EXPORTS_W double threshold(InputArray src, OutputArray dst,
                              double thresh, double maxval, int type );
```

该函数的第 5 个参数 type(阈值类型)是一个枚举类型，代码如下：

```
//chapter5/opencv4 - help - api.txt
enum ThresholdTypes {
    THRESH_BINARY = 0,
    THRESH_BINARY_INV = 1,
    THRESH_TRUNC = 2,
    THRESH_TOZERO = 3,
    THRESH_TOZERO_INV = 4,
    THRESH_MASK = 7,
    THRESH_OTSU = 8,
    THRESH_TRIANGLE = 16
};
```

打开上文创建的 MFC 工程(MFCOpenCV4Demo2)，在对话框设计界面中复制一份 Picture Control 控件，然后拖曳一个按钮，将 Caption 属性修改为“二值化”，如图 5-36 所示。双击该按钮会自动生成消息函数，代码如下：

```
//chapter5\MFCOpenCV4Demo2\MFCOpenCV4Demo2/MFCOpenCV4Demo2Dlg.cpp
void CMFCOpenCV4Demo2Dlg::OnBnClickedButton1(){
    //TODO: 在此添加控件通知处理程序代码
    //TODO: 在此添加控件通知处理程序代码
    Mat dst_mat;              //左边载入的图片对象

    //通过 imread ()函数读取本地图片
    Mat src_mat = imread("fyxylogo.png");
```

```
    //S1 表示灰度处理
    cv::cvtColor(src_mat, dst_mat, cv::COLOR_BGR2GRAY);

    //S2 表示开始二值化
    //第 2 个和第 3 个参数是阈值,决定了二值化显示的效果
    //如果设定不正确,则可能使图像显不出来
    cv::threshold(dst_mat, dst_mat, 127, 255,cv::THRESH_BINARY);

    //S3 表示展示图片处理后的效果
    DrawMat(dst_mat, IDC_STATIC_pic2);
}
```

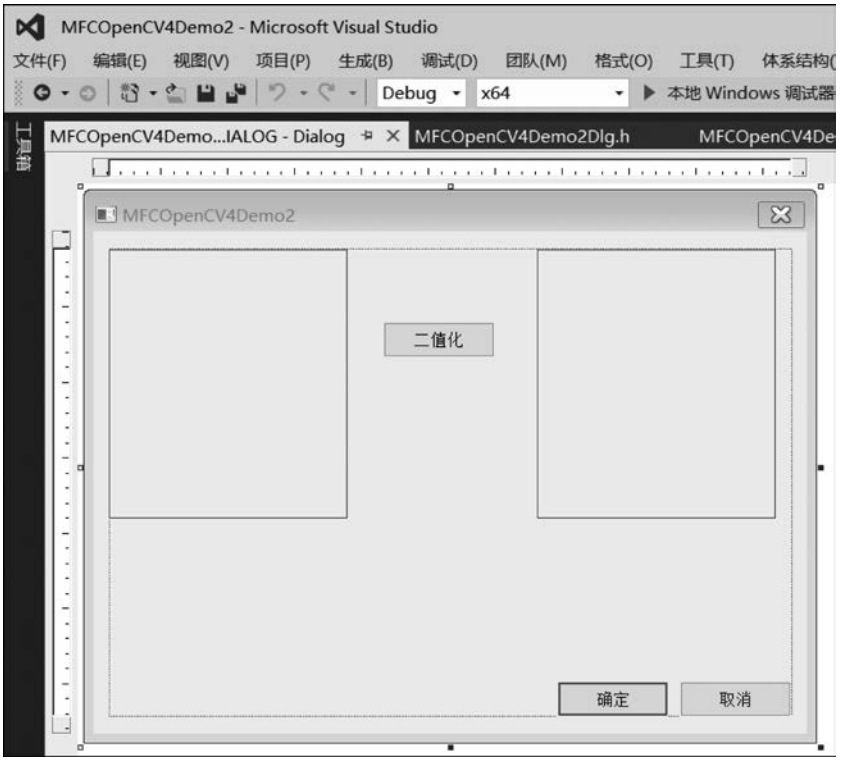


**图 5-36　MFC 二值化处理界面设计**

在该函数中,首先通过 imread()函数加载一张本地图片,接着调用 cv::cvtColor()函数进行灰度处理,然后调用 cv::threshold()函数进行二值化处理,最后将二值化处理后的 Mat 数据显示到右侧的 Picture Control 控件上,其中 DrawMat()函数是一个成员函数,用于将 Mat 数据显示到 MFC 的 Picture Control 控件上,代码如下:

```
//chapter5\MFCOpenCV4Demo2\MFCOpenCV4Demo2/MFCOpenCV4Demo2Dlg.cpp
//参数 1:要显示的图对象
//参数 2:Picture Control 控件的 ID
void CMFCOpenCV4Demo2Dlg::DrawMat(cv::Mat& img, UINT nID){
```

```
    CRect rect;
    cv::Mat imgTmp;

    GetDlgItem(nID)->GetClientRect(&rect);              //获取控件大小
    //缩放 Mat 并备份
    cv::resize(img, imgTmp, cv::Size(rect.Width(), rect.Height()));
    //再重新进行灰度处理,备用
    switch (imgTmp.channels()){
    case 1:
        cv::cvtColor(imgTmp, imgTmp, CV_GRAY2BGRA);      //GRAY 单通道
        break;
    case 3:
        cv::cvtColor(imgTmp, imgTmp, CV_BGR2BGRA);       //BGR 三通道
        break;
    default:
        break;
    }
    //计算一像素占用多少字节
    int pixelBytes = imgTmp.channels() * (imgTmp.depth() + 1);

    //制作 bitmapinfo(数据头)
    BITMAPINFO bitInfo;
    bitInfo.bmiHeader.biBitCount = 8 * pixelBytes;
    bitInfo.bmiHeader.biWidth = imgTmp.cols;
    bitInfo.bmiHeader.biHeight = -imgTmp.rows;
    bitInfo.bmiHeader.biPlanes = 1;
    bitInfo.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
    bitInfo.bmiHeader.biCompression = BI_RGB;
    bitInfo.bmiHeader.biClrImportant = 0;
    bitInfo.bmiHeader.biClrUsed = 0;
    bitInfo.bmiHeader.biSizeImage = 0;
    bitInfo.bmiHeader.biXPelsPerMeter = 0;
    bitInfo.bmiHeader.biYPelsPerMeter = 0;
    //Mat.data + bitmap 数据头 -> MFC
    CDC * pDC = GetDlgItem(nID)->GetDC();
    ::StretchDIBits(
        pDC->GetSafeHdc(),
        0, 0, rect.Width(), rect.Height(),
        0, 0, rect.Width(), rect.Height(),
        imgTmp.data,
        &bitInfo,
        DIB_RGB_COLORS,
        SRCCOPY
    );
    ReleaseDC(pDC);
}
```

该函数的主要工作是制作 BITMAPINFO 类型的图像头数据,然后结合 Mat 类型的原始图像数据,通过 StretchDIBits()函数显示到 Picture Control 控件上,其中 StretchDIBits()函数将 DIB 中矩形区域内像素使用的颜色数据复制到指定的目标矩形中；如果目标矩形比

源矩形大，则函数对颜色数据的行和列进行拉伸，以与目标矩形匹配；如果目标矩形比源矩形小，则该函数通过指定的光栅操作对行列进行压缩。该函数的声明代码如下：

```
//chapter5/opencv4 - help - api.txt
int StretchDIBits(HDC hdc, int XDest , int YDest , int nDestWidth, int nDestHeight, int XSrc,
int YSrc, int nSrcWidth, int nSrcHeight, CONST VOID * lpBits, CONST BITMAPINFO * lpBitsInfo,
UINT iUsage, DWORD dwRop);
```

如果函数执行成功，则返回值是复制的扫描线数目；如果函数执行失败，则返回值是GDI_ERROR。各个参数的含义如下。

(1) hdc：指向目标设备环境的句柄。

(2) XDest：指定目标矩形左上角位置的 $x$ 轴坐标，按逻辑单位表示坐标。

(3) YDest：指定目标矩形左上角的 $y$ 轴坐标，按逻辑单位表示坐标。

(4) nDestWidth：指定目标矩形的宽度。

(5) nDestHeight：指定目标矩形的高度。

(6) XSrc：指定 DIB 中源矩形(左上角)的 $x$ 轴坐标，坐标以像素表示。

(7) YSrc：指定 DIB 中源矩形(左上角)的 $y$ 轴坐标，坐标以像素表示。

(8) nSrcWidth：按像素指定 DIB 中源矩形的宽度。

(9) nSrcHeight：按像素指定 DIB 中源矩形的高度。

(10) lpBits：指向 DIB 位的指针，这些位的值按字节类型数组存储。

(11) lpBitsInfo：指向 BITMAPINFO 结构的指针，该结构包含关 DIB 方面的信息。

(12) iUsage：表示是否提供了 BITMAPINFO 结构中的成员 bmiColors，如果提供了，则该 bmiColors 是否包含了明确的 RGB 值或索引。参数 iUsage 必须取下列值，这些值的含义如下：

- DIB_PAL_COLORS：表示该数组包含对源设备环境的逻辑调色板进行索引的 16 位索引值。
- DIB_RGB_COLORS：表示该颜色表包含原义的 RGB 值。

(13) dwRop：指定源像素、目标设备环境的当前刷子和目标像素是如何组合形成新的图像的。

OpenCV 的二值化处理是一个常用的技术点，重新编译并运行该程序，效果如图 5-37 所示。

### 3. MFC+OpenCV 4 采集并显示摄像头

将 OpenCV 4 采集的摄像头数据显示到 MFC 的 Picture Control 控件上，与显示普通的图片几乎是一样的。这里将摄像头数据直接显示到左侧的 Picture Control 控件上，CMFCOpenCV4Demo2Dlg::OnInitDialog()函数中的代码不用变。在对话框设计界面上新拖曳一个按钮，将其 Caption 属性修改为“摄像头”，双击该按钮生成消息函数，修改后的代码如下：

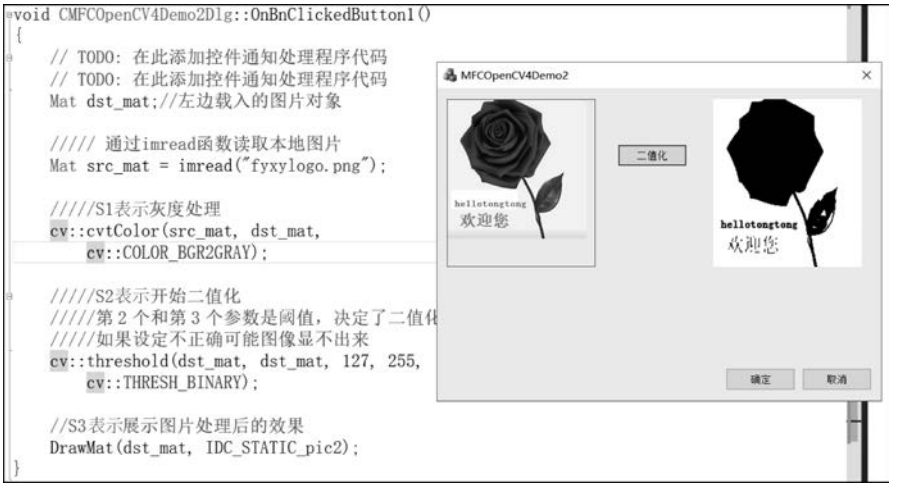


**图 5-37　OpenCV 4 二值化处理的运行效果**

```
//chapter5\MFCOpenCV4Demo2\MFCOpenCV4Demo2/MFCOpenCV4Demo2Dlg.cpp
void CMFCOpenCV4Demo2Dlg::OnBnClickedButtonCamera(){
    //TODO: 在此添加控件通知处理程序代码
    VideoCapture capture(0, cv::CAP_DSHOW);                    //打开摄像头
    cv::Mat frame;
    cv::Mat imgTmp;
    CRect rect;
    //获取 Picture 控件大小
    GetDlgItem(IDC_STATIC_pic1)->GetClientRect(&rect);

    while (true){
        capture.read(frame);                                   //读取摄像头一帧数据
        //缩放 Mat 并备份
        cv::resize(frame, imgTmp, cv::Size(rect.Width(), rect.Height()));
        imshow(OPENCV_WINDOW_NAME_PIC1, imgTmp);               //OpenCV 显示图片
        waitKey(33);
    }
}
```

重新编译并运行该程序，效果如图 5-38 所示。

### 4. VideoCapture 类详解

OpenCV 4 中从视频文件或摄像机中捕获视频的类是 VideoCapture。该类提供了 C++API，用于从摄像机捕获视频或读取视频文件。关于视频的读操作是通过 VideoCapture 类来完成的；视频的写操作是通过 VideoWriter 类实现的。VideoCapture 既支持从视频文件读取，也支持直接从摄像机(如计算机自带摄像头)中读取。如果想获取视频，则需要先创建一个 VideoCapture 对象，VideoCapture 对象的创建方式有以下 3 种。

1) 创建一个 VideoCapture 捕获对象，通过 open()成员函数来设定打开的内容

先创建 VideoCapture 对象，然后调用 open()成员函数来设定需要打开的内容，包括视


**图 5-38　MFC＋OpenCV 采集并显示摄像头**

频文件或图片等，案例代码如下：

```
//chapter5/VideoCaptureDemo.cpp
#include<iostream>
#include<opencv2/opencv.hpp>

using namespace cv;

int main(){
    VideoCapture capture;
    Mat frame;
    frame = capture.open("video2.mp4");
//frame = capture.open("water2.jpg");
    if (!capture.isOpened()){
        printf("can not open ...\n");
        return -1;
    }
    namedWindow("output", CV_WINDOW_AUTOSIZE);

    while (capture.read(frame)){
        imshow("output", frame);
        waitKey(60);
    }
    capture.release();
    return 0;
}
```

2）创建一个 VideoCapture 捕获对象，从摄像机中读取视频

先创建 VideoCapture 对象，然后调用 open()成员函数打开指定的摄像头。给 open()成员函数传递的参数是摄像头索引，从 0 开始，也可以通过第 2 个参数指定 CAP_DSHOW 方式，案例代码如下：

```
//chapter5/VideoCaptureDemo.cpp
#include<iostream>
#include<opencv2/opencv.hpp>
using namespace cv;

int main(){
    VideoCapture capture;
    capture.open(0);
    //capture.open(0, CAP_DSHOW);
    if (!capture.isOpened()){
        printf("can not open ...\n");
        return -1;
    }

    Size size = Size(capture.get(CV_CAP_PROP_FRAME_WIDTH), capture.get(CV_CAP_PROP_FRAME_
HEIGHT));
    VideoWriter writer; //存储到本地
    writer.open("D:/video3.avi", CV_FOURCC('M', 'J', 'P', 'G'), 10, size, true);

    Mat frame, gray;
    namedWindow("output", CV_WINDOW_AUTOSIZE);

    while (capture.read(frame)){
        //转换为黑白图像
        cvtColor(frame, gray, COLOR_BGR2GRAY);
        //二值化处理
    //  threshold(gray, gray, 0, 255, THRESH_BINARY | THRESH_OTSU);
        cvtColor(gray, gray, COLOR_GRAY2BGR);
    //  imshow("output", gray);
        imshow("output", frame);
        writer.write(gray);
        waitKey(10);
    }

    waitKey(0);
    capture.release();
    return 0;
}
```

3）创建一个 VideoCapture 捕获对象，从文件中读取视频

先创建 VideoCapture 对象，然后调用 open()成员函数打开指定的视频文件。给 open()成员函数传递的参数是文件的完整路径，案例代码如下：

```
//chapter5/VideoCaptureDemo.cpp
#include<iostream>
#include<opencv2/opencv.hpp>
using namespace cv;

int main(){
    VideoCapture capture;
    Mat frame;
    frame = capture("D:\\video3.avi");
    if (!capture.isOpened()){
        printf("can not open ...\n");
        return -1;
    }
    namedWindow("output", CV_WINDOW_AUTOSIZE);

    while (capture.read(frame)){
        imshow("output", frame);
        waitKey(60);
    }
    capture.release();
    return 0;
}
```

### 5. VideoWriter 类详解

OpenCV 4 提供了 VideoWriter 类，用于保存视频，只支持保存.avi 格式的视频，保存的视频目前无法避免被压缩，而且不能添加音频。VideoWriter 类的构造函数的原型如下：

```
VideoWriter(const String& filename, int fourcc, double fps,
            Size frameSize, bool isColor = true);
```

该函数的各个参数的含义如下。

(1) filename：输出视频的文件名。

(2) fourcc：使用 4 个字符表示压缩帧的编解码格式(Codec)，常见格式如下：

```
//chapter5/VideoCaptureDemo.cpp
CV_FOURCC('M','J','P','G') motion-jpeg codec
CV_FOURCC('P','I','M','1') MPEG-1 codec
CV_FOURCC('M','J','P','G') motion-jpeg codec
CV_FOURCC('M', 'P', '4', '2') MPEG-4.2 codec
CV_FOURCC('D', 'I', 'V', '3') MPEG-4.3 codec
CV_FOURCC('D', 'I', 'V', 'X') MPEG-4 codec
CV_FOURCC('U', '2', '6', '3') H263 codec
CV_FOURCC('I', '2', '6', '3') H263I codec
CV_FOURCC('F', 'L', 'V', '1') FLV1 codec
```

(3) fps：输出视频的帧率。

(4) frameSize：输出视频的宽和高。

(5) isColor：将输出的视频设置为彩色或者灰度。

**注意**：OpenCV 是一个视觉库，主要用于处理计算机视觉，但它并不擅长处理视频编解码。FFmpeg 是专门用来处理视频编解码的库，支持硬件加速，功能强大。建议专业的视频编解码功能使用 FFmpeg 开源库。

1) 修改视频的分辨率

先创建 VideoCapture 对象，然后调用它的 set()函数修改视频的相关属性，这些属性包括视频宽度(CAP_PROP_FRAME_WIDTH)、视频高度(CAP_PROP_FRAME_HEIGHT)和视频帧率(CAP_PROP_FPS)等，然后创建 VideoWriter 对象，存储目标视频，代码如下：

```
//chapter5/VideoCaptureDemo.cpp
#include<opencv2/opencv.hpp>
#include<iostream>
using namespace std;
using namespace cv;

int main(){
    //获取视频,需要根据自己的视频路径进行修改
    VideoCapture capture("./left_02.mp4");
    if (!capture.isOpened())
        return -1;
    Mat frame;
    //修改视频的宽度和高度
    capture.set(CAP_PROP_FRAME_WIDTH, 640);
    capture.set(CAP_PROP_FRAME_HEIGHT, 480);
    Mat image;
    //VideoWriter(const String &filename, int fourcc, double fps,Size frameSize, bool isColor
= true);
    //将图像的帧速修改为 30,图像帧的大小是(640,480)
    VideoWriter videowriter("./output/result11.avi",
        VideoWriter::fourcc('M', 'J', 'P', 'G'), 30, Size(640, 480), true);

    while (capture.read(image)){
        imshow("image", image);
        resize(image, image, Size(640, 480), INTER_LINEAR);
        videowriter.write(image); //逐帧写入
        waitKey(1);
    }
    waitKey();
    return 0;
}
```

2) 在视频的指定区域画圆

直接创建 VideoWriter 对象，通过 imread()函数读取图片，然后调用 cv::putText()函数和 cv::circle()函数分别绘制文字和圆形，最后写入视频文件中。编译并运行该程序，效果如图 5-39 所示，案例代码如下：

```
//chapter5/VideoCaptureDemo.cpp
#include <opencv2/opencv.hpp>
#include <iostream>

int octest_imgCircle(){
    cv::Size image_size(640, 480);
    std::string outputVideoPath = "./image_save.avi";

    cv::VideoWriter outputVideo;
    outputVideo.open(outputVideoPath,
        VideoWriter::fourcc('M', 'P', '4', '2'),
        20.0, image_size);
    Mat img;
    std::vector<string> imagelist;
    imagelist.push_back("opencv.bmp");
    //imagelist.push_back("opencv.png");

    std::cout << std::endl << "---Begin----" << std::endl;
    for (int i = 0; i < imagelist.size(); i++){
        //Read images
        img = cv::imread(imagelist[i]);
        //for show
        cv::putText(img, "(100,100)", cv::Point2f(100, 100), 1, 1, cv::Scalar(255, 0, 0), 1);
        cv::circle(img, cv::Point2f(100, 100), 50, cv::Scalar(255, 0, 0), 1);

        outputVideo << img;
        cv::imshow("img", img);
        cv::waitKey(1);
    }
    img.release();
    return 0;
}
int main(){
    return octest_imgCircle();
}
```

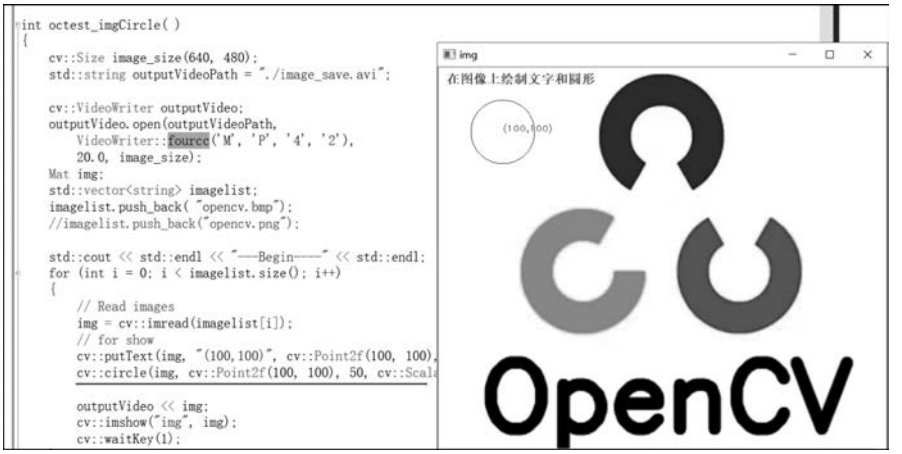


图 5-39　OpenCV 在图像指定区域画圆

3）对彩色图像的每个通道单独进行处理

调用 split()函数可以将彩色视频分离出 3 个通道进行单独处理，然后可以调用 merge()函数进行通道合并，案例代码如下：

**注意**：OpenCV 中的颜色通道顺序不是 RGB，而是 BGR。

```
//chapter5/VideoCaptureDemo.cpp
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace cv;
using namespace std;

int main(){
    //cap1 是左边镜头，cap2 是右边镜头
    VideoCapture cap1("./left_02.mp4");
    VideoCapture cap2("./right_02.mp4");

    double rate = 60;
    int delay = 1000 / rate;
    bool stop(false);
    Mat frame1; Mat frame2; Mat frame;
    Point2i a; //存储偏移量

    //将图像的帧速修改为 30，图像帧的大小是(1920,1080)
    VideoWriter videowriter("./result356.avi",
        VideoWriter::fourcc('M', 'J', 'P', 'G'), 30, Size(1920, 1080), true);

    if (cap1.isOpened() && cap2.isOpened()){
        cout << "***  ***" << endl;
        cout << "打开成功!" << endl;
    }
    else{
        cout << "***  ***" << endl;
        cout << "警告:打开不成功或者未检测到有两个视频!" << endl;
        cout << "程序结束!" << endl << "***  ***" << endl;
        return -1;
    }
    Mat image, image2;
    while (!stop){
        if (cap1.read(frame1) && cap2.read(frame2)){
            imshow("cam1", frame1);
            imshow("cam2", frame2);

            //彩色帧转灰度
            //cvtColor(frame1, frame1, COLOR_RGB2GRAY);
```

```
                //cvtColor(frame2, frame2, COLOR_RGB2GRAY);
                //imshow("cvtColor1", frame1);
                //imshow("cvtColor2", frame2);
                image = frame1;

                Mat src = image;
                Mat res(src.rows, src.cols, CV_8UC3); //用来存储目的图片的矩阵
                imshow("src", src);

                //Mat 数组用来存储分离后的 3 个通道,每个通道都被初始化为 0
                //MATLAB 的排列顺序是 R、G、B,而在 OpenCV 中,排列顺序是 B、G、R
                Mat planes[] = {
                    Mat::zeros(src.size(), CV_8UC1),
                    Mat::zeros(src.size(), CV_8UC1),
                    Mat::zeros(src.size(), CV_8UC1) };
                //多通道分成 3 个单通道
                //在 OpenCV 中,一张三通道图像的一像素是按 BGR 的顺序存储的
                //可以通过 planes[0]、planes[1]和 planes[2]分别对每个通道进行处理
                split(src, planes);
                merge(planes, 1, res); //通道合并,三通道合并为一张完整的彩色图片
                imshow("name", res);
                waitKey(1);
            }
        }

        return 0;
    }
```

## 5.4 MFC 结合 OpenCV 实现采集和录制功能

本节使用 OpenCV 结合 MFC 实现采集和录制功能,同时也实现了拍照功能,程序运行起来之后单击“打开摄像头”就可以预览到摄像头采集的画面,如图 5-40 所示,然后单击“单击拍照”按钮,此时会弹出拍照界面及相关的操作按钮(如灰度、变亮、变暗等),如图 5-41 所示。

**注意**:本节案例的完整工程代码可参考本书源码中的 chapter5/OpenCVCameraDemo3,建议读者先下载源码将工程运行起来,然后结合本书进行学习。

### 1. 项目结构及界面设计

双击 OpenCVCameraDemo3.sln 会打开整个工程(笔者本地安装的是 VS 2015,读者也可以使用更高的 VS 版本),如图 5-42 所示。由于官方发布的 OpenCV 4 开发库只支持 64 位,所以这里选择 x64 开发模式。程序界面主要包括两个对话框,即主对话框界面和拍照对话框界面,其中主对话框包括上方的 Picture Control 控件和 4 个按钮(打开摄像头、关闭摄


图 5-40　MFC＋OpenCV 采集并录制摄像头


图 5-41　MFC＋OpenCV 拍照功能

像头、单击拍照和开启摄像），拍照对话框包括左侧的 Picture Control 控件和 7 个按钮（灰度、恢复、变亮、变暗、对比度＋、对比度－和保存图片）。

由于该项目用到了 OpenCV 4，所以需要配置包含目录、库目录和附加依赖项（详细步骤可参考“5.1 VS 2015 搭建 OpenCV 4 开发环境”节），如图 5-43 所示。

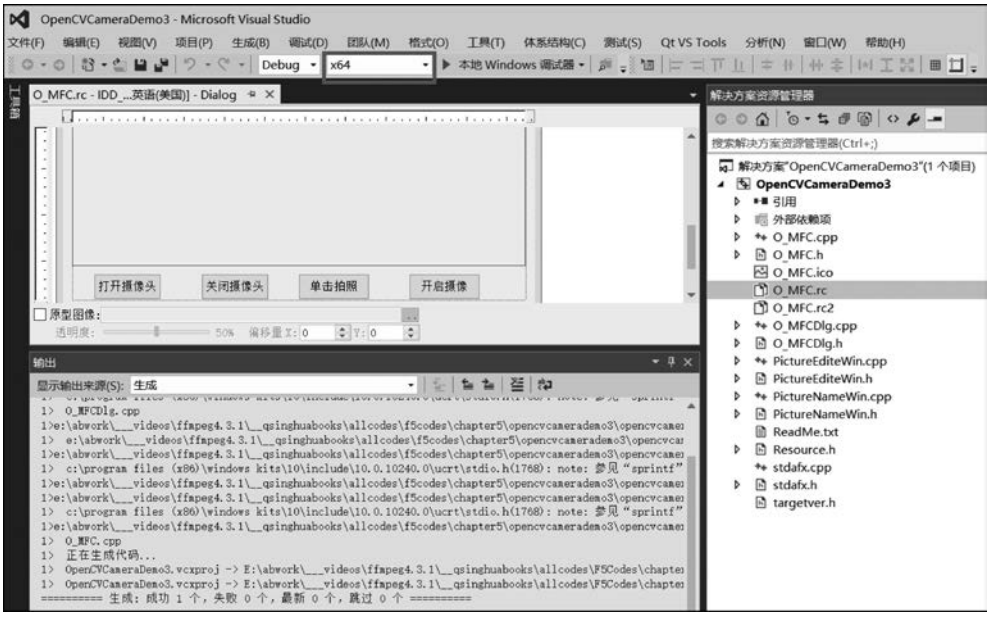


图 5-42 OpenCVCameraDemo3 项目工程

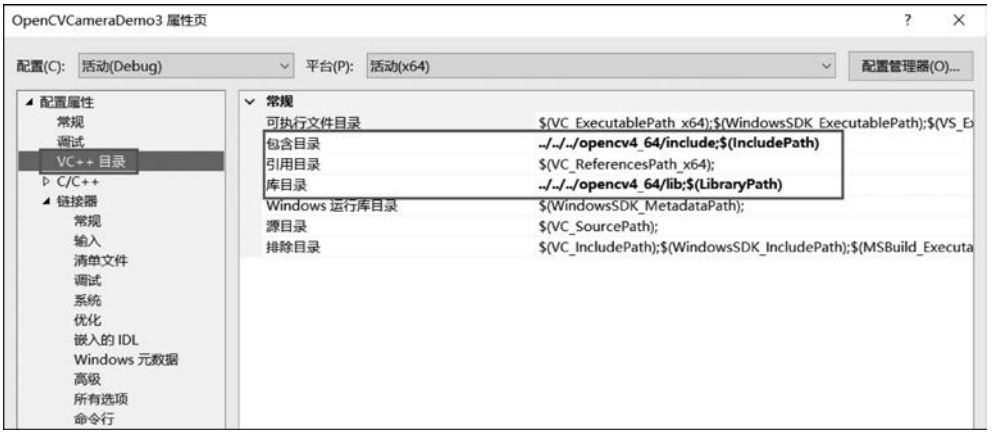


图 5-43 配置 OpenCV 的包含目录和库目录

**注意**：读者可以打开本地的 VS 2015/2017/2019，新建一个基于对话框的 MFC 应用程序，在主对话框中拖曳 1 个 Picture Control 控件和 4 个按钮，然后新增一个对话框，用于实现拍照功能，拖曳 1 个 Picture Control 控件和 7 个按钮。修改这些按钮的 Caption 属性，然后分别双击这些按钮，生成对应的消息函数。

## 2. 主对话框类

该项目的主对话框类是 CO_MFCDlg，继承自 CDialogEx，主要使用 VideoCapture 和

VideoWriter这两个类来分别实现摄像头的采集和录制功能，该类的头文件代码如下（详情可参考注释信息）：

```
//chapter5/OpenCVCameraDemo3/OpenCVCameraDemo3/O_MFCDlg.h
//O_MFCDlg.h : header file
#include "PictureNameWin.h"
#include "PictureEditWin.h"
#include <cstring>
#include <string.h>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#pragma once
using namespace std;

//CO_MFCDlg dialog
class CO_MFCDlg : public CDialogEx{
//Construction
public:
    CO_MFCDlg(CWnd* pParent = NULL);              //standard constructor
    //将 OpenCV 的显示窗口与 MFC 的控件连接起来
    bool attachWindow(string &pic,const char* name,int ID);
    //效果是 OpenCV 的窗口恰好覆盖在控件上
    bool showImage(string pic,int id,cv::Mat mat);
    CString GetModuleDir();
    char* CStringToChar(CString pstr);
//Dialog Data
    enum { IDD = IDD_O_MFC_DIALOG };

    protected:
    virtual void DoDataExchange(CDataExchange* pDX);//DDX/DDV support

//Implementation
protected:
    HICON m_hIcon;

    //Generated message map functions
    virtual BOOL OnInitDialog();
    afx_msg void OnClose();
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnTimer(UINT_PTR nIDEvent);
    afx_msg void OnBnClickedButton_close();        //关闭摄像头
    afx_msg void OnBnClickedButton_open();         //打开摄像头
    afx_msg void OnBnClickedButton_snap();         //拍照
    afx_msg void OnBnClickedButton_store();        //录制并存储
    virtual BOOL PreTranslateMessage(MSG* pMsg);
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    DECLARE_MESSAGE_MAP()
public:
```

```
    cv::VideoCapture * video;              //摄像头捕获
    string videowin_pic;
    bool vedio_flag;                       //摄像开始标记
    bool open_flag;
    char curdir[128];                      //保存应用程序当前所在的路径
    char picpath[128];                     //保存默认图片的路径
    char vediopath[128];                   //保存视频的路径
    cv::Mat mat;                           //保存抓拍的每帧图片
    cv::Mat t_mat;                         //保存拍照的图片

    CButton m_vedioctl;
    VideoWriter * videowrite;              //视频存储
};
```

### 3. 对话框初始化函数

对话框的 CO_MFCDlg::OnInitDialog()初始化函数主要用于完成各个成员变量的初始化工作,需要将 opencv.png 图片复制到.exe 文件所在路径,否则运行时会报错,其中 attachWindow()函数用于绑定 OpenCV 与 MFC 的 IDC_STATIC 控件,这样就可以将 OpenCV 单独弹出来的窗口嵌入 MFC 的控件上。这两个函数的主要代码如下:

```
//chapter5/OpenCVCameraDemo3/OpenCVCameraDemo3/O_MFCDlg.cpp
bool CO_MFCDlg::attachWindow(string &pic,const char * name,int ID){
    pic = string(name);
    cv::namedWindow(pic, 1);                          //创建 OpenCV 窗口
    HWND hWnd = (HWND) cvGetWindowHandle(name);       //获取 OpenCV 的窗口句柄
    HWND hParent = ::GetParent(hWnd);                 //获取 OpenCV 窗口的父窗口句柄
    ::ShowWindow(hParent, SW_HIDE);                   //隐藏原父窗口
    ::SetParent(hWnd, GetDlgItem(ID)->m_hWnd);        //将新父窗口设置为 MFC 的控件
    return true;
}

BOOL CO_MFCDlg::OnInitDialog(){
    CDialogEx::OnInitDialog();
    //...省略代码

    //TODO: Add extra initialization here IDD_O_MFC_DIALOG
    this->open_flag = false;
    this->vedio_flag = false;
    this->video = NULL;
    this->videowrite = NULL;
    memset(this->curdir,0,sizeof(this->curdir));
    memset(this->vediopath,0,sizeof(vediopath));
    CString msg= this->GetModuleDir();
    char * tt_curdir = this->CStringToChar(msg);
    memcpy(this->curdir,tt_curdir,strlen(tt_curdir));
    delete tt_curdir;
    tt_curdir=NULL;
    memset(picpath,0,sizeof(picpath));
```

```
    sprintf(picpath,"opencv.png",this->curdir);

    this->attachWindow(videowin_pic,"camera",IDC_STATIC);
    cv::Mat mat = cv::imread(picpath);
    showImage(this->videowin_pic,IDC_STATIC,mat);

    return TRUE; //return TRUE unless you set the focus to a control
}
```

其中，showImage()函数是封装的一个私有函数，用于将指定的 Mat 图片，根据 IDC_STATIC 控件来调整宽和高，然后显示出来，代码如下：

```
//chapter5/OpenCVCameraDemo3/OpenCVCameraDemo3/O_MFCDlg.cpp
bool CO_MFCDlg::showImage(string pic,int id,cv::Mat mat){
    CRect rect;
    GetDlgItem(id)->GetClientRect(&rect);
cv::resize(mat,mat,cv::Size(rect.Width(),rect.Height()),CV_INTER_CUBIC);

    imshow(pic,mat);
    return true;
}
```

### 4. 打开摄像头

首先构造出 VideoCapture 对象，并指定 cv::CAP_DSHOW 方式，然后调用 AfxBeginThread()函数开启一条独立的线程用来循环显示摄像头捕获的图像，代码如下：

```
//chapter5/OpenCVCameraDemo3/OpenCVCameraDemo3/O_MFCDlg.cpp
UINT CaptureThread(LPVOID * aPram) {                    //摄像头抓拍线程
    CO_MFCDlg* dlg = (CO_MFCDlg*)aPram;

    while(dlg->open_flag){
        (*dlg->video) >> dlg->mat;                      //该功能用于视频录制,可以省略
        if(dlg->vedio_flag){
            DrawEllipse(dlg->mat,0);                    //摄像时绘制小红圈
            if (dlg->videowrite) {                      //写入视频帧
                dlg->videowrite->write(dlg->mat);
            }
        }
        dlg->showImage(dlg->videowin_pic,IDC_STATIC,dlg->mat);
        cv::waitKey(30);
    }
    return 0;
}

void CO_MFCDlg::OnBnClickedButton_open() {              //打开摄像头
    //TODO: Add your control notification handler code here
    this->open_flag=true;
    video = new cv::VideoCapture(0, cv::CAP_DSHOW);
    if(!video)
```

```
        return ;
    AfxBeginThread((AFX_THREADPROC)CaptureThread,this);
}
```

### 5. 关闭摄像头

首先将 open_flag 成员变量设置为 false，然后显示初始化图片（opencv. png），代码如下：

```
//chapter5/OpenCVCameraDemo3/OpenCVCameraDemo3/O_MFCDlg.cpp
void CO_MFCDlg::OnBnClickedButton_close(){                    //关闭摄像头
    //TODO: Add your control notification handler code here
    this->open_flag = false;
    cv::Mat mat = cv::imread(picpath);
    showImage(this->videowin_pic,IDC_STATIC,mat);
}
```

### 6. 拍照

拍照功能是通过 PictureEditWin 这个对话框类来完成的，代码如下：

```
//chapter5/OpenCVCameraDemo3/OpenCVCameraDemo3/O_MFCDlg.cpp
void CO_MFCDlg::OnBnClickedButton_snap() {          //单击拍照
    //TODO: Add your control notification handler code here
    if(this->open_flag){
        this->t_mat = this->mat;
        PictureEditWin pewin;
        cv::Mat tt_mat = this->t_mat.clone();
        pewin.setMat(tt_mat);
        pewin.DoModal();
    }
    else{
        this->MessageBox(_T("摄像头未开启!"));
    }
}
```

PictureEditWin 类对传递进来的 Mat 图片数据进行各种处理，包括变亮、变暗等，这些按钮的功能比较单一，例如变亮功能的代码如下（其他按钮的代码不再赘述）：

```
//chapter5/OpenCVCameraDemo3/OpenCVCameraDemo3/PictureEditWin.cpp
void PictureEditWin::OnBnClickedButton2(){          //变亮按钮处理事件
    //TODO: Add your control notification handler code here
    for(int y = 0;y < tempImage.rows;++y){          //总行数
        for(int x = 0;x < tempImage.cols;++x){      //总列数
            for(int c = 0;c < 3;++c){               //分别对每个像素的 3 个通道进行处理
        //beta 是成员变量，初始化为 5，用来修改亮度. 变亮时每次加 5，变暗时每次减 5
        tempImage.at < cv::Vec3b >(y,x)[c] =
        cv::saturate_cast < uchar >(tempImage.at < cv::Vec3b >(y,x)[c] + beta);
            }
```

```
        }
    }
    showImage(pic,IDC_STATIC,tempImage);
    this->UpdateData(false);
}
```

分析该函数代码会发现，对每个像素的3个通道进行单独处理，其中saturate_cast()函数是防溢出保护，参数为uchar时大致原理的伪代码如下：

```
if(data<0) data=0;
else if(data>255) data=255;
```

cv::staturate_cast()函数主要用来对计算结果进行截断，截断的结果范围为0～255，同理可以用作其他类型限定值的范围。

**7. 录制**

单击界面上的“开启摄像”按钮，就会构造出VideoWriter对象，并将vedio_flag标志值设置为true，真正的逐帧录制工作在这个CaptureThread()线程函数中，代码如下：

```
//chapter5/OpenCVCameraDemo3/OpenCVCameraDemo3/O_MFCDlg.cpp
UINT CaptureThread(LPVOID * aPram){                    //摄像头抓拍线程
    CO_MFCDlg* dlg=(CO_MFCDlg*)aPram;

    while(dlg->open_flag){
        (*dlg->video)>>dlg->mat;
        if(dlg->vedio_flag){
            DrawEllipse(dlg->mat,0);                   //摄像时绘制小红圈，可以省略
            if (dlg->videowrite) {
                dlg->videowrite->write(dlg->mat);
            }
        }
        dlg->showImage(dlg->videowin_pic,IDC_STATIC,dlg->mat);
        cv::waitKey(30);
    }
    return 0;
}

void CO_MFCDlg::OnBnClickedButton_store() {            //开始摄像
    //TODO: Add your control notification handler code here
    if(this->open_flag && !this->vedio_flag){
        //弹出输入名称对话框
        PictureNameWin picwin;                         //先弹出对话框，输入存储文件名
        picwin.setWtype(2);
        picwin.DoModal();
        if(!picwin.filename){return ;}
        char t_filepath[128];
        memset(t_filepath,0,sizeof(t_filepath));
        sprintf(t_filepath,".\\%s",picwin.filename);
        memcpy(vediopath,t_filepath,sizeof(t_filepath));
```

```
            m_vedioctl.SetWindowText(_T("结束摄像"));
            ///下面构造 VideoWriter 对象,并设置标志值
            this->videowrite = new VideoWriter(vediopath,
                VideoWriter::fourcc('X', 'V', 'I', 'D'),
                25, cvSize(this->mat.cols, this->mat.rows),CAP_DSHOW);
            //保存的文件名,编码为 XVID,
            //大小就是摄像头视频的大小,帧频率是 25
            if(video) //如果能创建 CvVideoWriter 对象,则表明成功
            { cout<<"VideoWriter has created."<< endl; }
            this->vedio_flag = true;
        }
        else if(this->open_flag && this->vedio_flag){
            m_vedioctl.SetWindowText(_T("开启摄像"));
            this->vedio_flag = false;
            this->videowrite->release();
            videowrite = NULL;
        }
        else{
            this->MessageBox(_T("摄像头未开启!"));
        }
        this->UpdateData(false);
    }
```