第3章 Video I/O 与 GUI

上一章中介绍了图像的基本知识和图像的基本操作，这一章将介绍 OpenCV 的图形用户接口（Graphical User Interface）模块 highgui。higui 模块主要目的是快速测试算法的结果、中间步骤的输出、简单的用户交互响应等，其主要功能如下：

- 创建图像显示窗口，支持简单的窗口操作功能
- 支持滚动条（trackbar）控制，处理简单的鼠标和键盘事件
- **从磁盘或内存读写图像**
- 读取视频流或者视频文件，保存视频文件

在本章中，我们将学习 highgui 的以下功能：

- 程序运行过程中获取并响应键盘输入
- 获取鼠标事件并相响应，完成回调功能函数与执行，实现鼠标控制程序
- 使用滚动条改变运行程序输入参数
- 如何从命令行输入参数执行程序

3.1Video I/O 术语

3.1.1 视频分辨率

如果你经常上视频网站或者有使用视频软件的经历，一定知道几个常用的视频分辨率（video resolution）：

- 标准分辨率（SD-Standard Definition）
- 高分辨率（HD-High Definition）
- 超高清分辨率（UHD- Ultra High Definition 或 4K）

这些分辨率是根据视频帧的大小（宽 x 高）来定义的，具体如表 3.1 所示。

表 3.1 分辨率定义

分辨率	帧大小
SD	640x360 或 720x480
HD	1280x720(720P)
Full HD	1920x1080(1080P)
UHD	3840x2160(4K)

3.1.2 帧率

一段视频实际上是播放一组图像而产生。图像的播放速度即多长时间播放一张图像，可

用帧率（FPS, Frame per Second）来衡量。帧率的意思是每秒钟播放的连续图像的数量（帧数）。如果一个视频的 FPS 为 40，意味着每秒钟播放 40 帧（40 张图像），即每间隔 25 毫秒播放一帧图像。

3.1.3 常用的视频格式

下面简单介绍一下 OpenCV 中写视频文件最常用的两种视频格式：AVI 和 MP4。

AVI

AVI（Audio Video Interleave）格式是 1992 年由微软提出的多媒体容器格式。AVI 格式把音频与视频放在同一个文件容器中，支持音视频同步回放，可以包含多个音频流和多视频流。AVI 格式是基于资源交互换文件格式(RIFF – Resource Interchange File Format)开发的， RIFF 这种容器格式用于存储多媒体数据。由于 AVI 格式较其它格式压缩得少一点，所以 AVI 文件会相对较大。

MP4

MP4 是动态图像专家组（Moving Picture Experts Group– MPEG）制定的一种格式，它是 MPEG-4 的缩写。MP4 是一种用于存储音频、视频或音视频的现代压缩格式。MP4 由二十几个的子标准组成，这些子标准称为部（part），第二部分（mp4 part 2）是最常提及的。

MP4 原本设计是使用最小的比特率进行压缩，但是随着大公司对 MP4 的使用的增加，比特率得到提升，现在的 MP4 都是高质量的。由于压缩降低了屏幕分辨率，有效的减少了内存使用，这使得 MP4 成为最常使用的格式，从大学演示文稿里的 MP4 格式视频到公司广告宣传片都使用 MP4 格式，同时它也可以在各种播放平台上播放。

AVI 与 MP4 一个最大的不同是 AVI 不支持 HECV/H.265 或者 VP9 格式。HECV/VP9 是超清视频编解码。MP4 支持这两种编解码，所以可以播放 4K UHD，而 AVI 只能播放 HD 和 FullHD。另外，AVI 文件本身不能提供字幕，需要 SubRip、SubStation Alpha 与 XSUB 等第三方字幕文件。

3.2 OpenCV 中显示相关的函数

cv::namedWindow

创建一个指定名称的窗口函数，第一个输入参数是窗口名称，第二输入参数控制窗口的是否可以被 resize。默认情况下窗口自动设置大小(WINDOW_AUTOSIZE)，表示窗口大小自动根据图像大小匹配，WINDOW_NORMAL 表示窗口大小可以手动 resize。函数解释如下：

```
01 void cv::namedWindow(
02     const String &  winname, // 窗口名称
03     int flags = WINDOW_AUTOSIZE // 默认大小不可修改
04 )
```

其中：

Winname 表示窗口名称，名称是窗口的唯一标识。

Flags 表示窗口标志，不同标志代表窗口的不同属性

cv::waitKey

该函数在 OpenCV 中使用场景非常多，从图像处理到视频处理都有应用，waitKey 是一个键盘绑定功能，会响应用户键盘操作，唯一输入参数是毫秒数，表示在等待的毫秒时间内响应键盘操作，如果用户没有键盘输入，等待的毫秒数结束程序继续执行；如果参数为 0 表示用户按下任意一个键，程序继续执行，否则就会一直阻塞等待。

```
int cv::waitKey(int delay = 0)
```

其中：

delay 表示等待时间，默认为 0

cv::destroyWindow

该函数销毁或者关闭指定的窗口，需要提供的唯一输入参数是窗口名称

```
void cv::destroyWindow(const String & winname)
```

其中：

winname 表示需要关闭或者销毁的窗口名称

cv::destroyAllWindows

该函数关闭或者销毁所有窗口，不需要输入参数。

```
void cv::destroyAllWindows()
```

3.3OpenCV 中视频读取与显示

就如 imread 可以读取图像一样，读取视频我们使用 VideoCapture 对象，首先创建一个 VideoCapture 实例，然后调用 read 方法即可读取视频。语法如下：

```
cv::VideoCapture::VideoCapture(
    const String &  filename,
    intapiPreference = CAP_ANY
)
```

其中

参数 filename 可以是

a) 视频文件
b) 图像序列文件的则表达(eg,image_%02.jpg，则会读取 image_00.jpg, image_01.jpg…..)
c) 视频 URL 地址，为视频流或者 IP 相机地址

apiPreference 表示后台视频编解码用的是哪个第三方库，常见支持有 cv::CAP_FFMPEG or cv::CAP_IMAGES or cv::CAP_DSHOW。

创建视频 Reader 对象

```
VideoCapturecap(args)
```

最常见的有下面三种读取方式

1. 从 webcam 中读取 args=0
2. 从视频文件中读取 args 为文件路径
3. 从图像序列中读取 args 可以为 image_%02.jpg

注意 webcam

多数时候只有一个相机链接到系统，因此我们设置输入参数 args=0。OpenCV 链接相机的时

候，系统有多个相机，我们需要打开第二个相机设置输入参数 args =1，打开第三个相机时候设置 args=2，如果更多，以此类推即可。

打开视频文件，首先导入需要的头文件

```
#include"opencv2/opencv.hpp"
#include <iostream>
using namespacestd;
using namespacecv;
```

根据视频路径，打开视频文件

```
string base_dir="D:/opencv_4.1.2/opencv/sources/samples/data/";
VideoCapturecap(base_dir+"vtest.avi");
// 检查是否打开
if(!cap.isOpened()) {
    cout<<"Error opening video stream or file"<<endl;
}
```

读取视频内容

如果上述代码执行正确就打开了视频文件，下面就读取视频内容，读取的方式是按帧读取，通过 cap.read 方式读取。

显示视频

使用 waitKey 与 imshow 一起来显示视频，其中 waitKey 实现每帧之间的暂停，这里 waitKey 函数输入参数是一个大于 0 的毫秒数值，因为 0 表示阻塞，会一直等待下去；而视频显示需要连续指定时间间隔不断显示读取到的视频帧画面；所以 waitKey 需要一个大于 0 的值。这个值是一个决定视频播放速度的毫秒数。

waitKey 与 Webcam

使用 Webcam 的时候因为帧率的限制，我们设置 waitKey 只需要暂停 1 毫秒，即 waitKey(1)

waitKey 与视频文件

当读取视频文件进行处理时候，我们通常也会设置 waitKey(1)，这样做可以让程序有足够空闲时间来对读取的 frame 进行处理后显示。

除非有必要，一般情况下都不会把 waitKey 的等待时间设置为大于 1。

读取与显示视频的框架代码显示如下：

```
// 循环读取视频
while(cap.isOpened())
{

    Mat frame;

    // read frame
    cap >> frame;

    //退出如果为空
    if (frame.empty())
        break;

    // 显示frame
    imshow("Frame", frame);

    // 阻塞等待25毫秒之后读取下一帧
    // 放慢播放速度
    waitKey(25);
}
```

3.4 VideoCapture 视频属性

我们已经学习了视频读取与显示，本节我们将学习到如何通过 VideoCapture 查询与设置视频属性，相关的两个方法是 get 与 set，具体如下：

cap.get(propId)

cap.set(propId, value)

其中 cap 是 VideoCapture 对象实例，propId 表示属性 id 值，value 表示我们想对当前属性 id 设置的值。常见的视频属性列表如下：

属性-枚举类型	值	解释
CAP_PROP_POS_MSEC	0	视频文件当前帧位置对应毫秒数
CAP_PROP_FRAME_WIDTH	3	视频帧宽度-width
CAP_PROP_FRAME_HEIGHT	4	视频帧高度-height
CAP_PROP_FPS	5	帧率
CAP_PROP_FOURCC	6	编解码方式
CAP_PROP_FRAME_COUNT	7	视频文件总帧数

查询视频属性

通过调用 get 方法输入指定属性 id 就可以查询指定的视频属性，查询视频帧的宽高，代码如下：

```
VideoCapturecap(base_dir+"vtest.avi");
int width = cap2.get(CAP_PROP_FRAME_WIDTH);
int height = cap2.get(CAP_PROP_FRAME_HEIGHT);
```

设置视频属性

通过 set 方法可以设置视频属性，这里设置的是跟视频设备相关的属性，也可以设置读取到的视频帧的宽高。注意：有时候 set 不一定得到你期望的结果，可能是因为你读取的是视频文件，这些属性无法更改；也可能是因为 webcam 本身并不支持这些属性设置。比如，对 Webcam 来说，假设默认分辨率是 720x1280，想重新设置为 200x200，很遗憾不支持，只能

改到 640x480，这是因为多数 Webcam 只会支持这个分辨率输出。设置视频属性代码演示如下：

```
// Set position of video to 2.5 seconds
cap2.set(CAP_PROP_POS_MSEC, 2500);

// Width
cap2.set(CAP_PROP_FRAME_WIDTH, 320);

// Height
cap2.set(CAP_PROP_FRAME_HEIGHT, 180);
```

3.5 写视频

在视频读取显示与处理之后，下一步我们想做的也许是保存视频。对图像来说，我们可以直接调用 imwrite 提供保存路径与 Mat 对象就可以实现保存。但是视频保存要求我们提供更多的信息才可以。它的大致步骤如下：

1.创建 VideoWriter 对象

```
cv::VideoWriter::VideoWriter(
    const String & filename,
    intfourcc,
    double  fps,
    Size    frameSize,
    boolisColor = true
)
```

参数解释

filename 表示保存的视频文件名称

fourcc 表示视频编码格式

fps 表示保存视频文件的帧率

frameSize 表示视频帧大小

isColor 不为 0 表示彩色编码，否则为灰度(只是在 windows 系统上支持)

2.通过 write 方法循环写入(调用 write 方法)

3.关闭释放对象(调用 release 方法)

FourCC Code

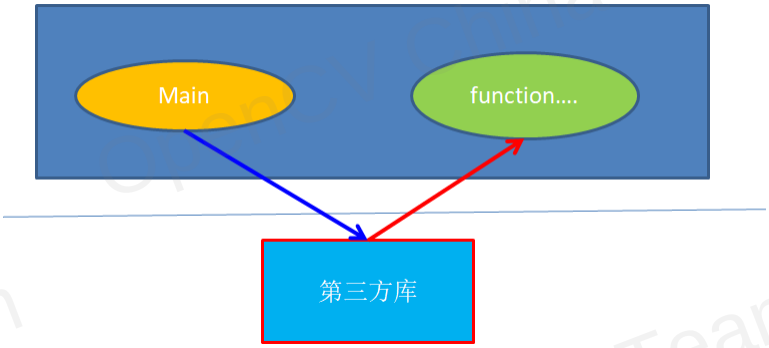
四字节的编码来表示视频编解码方式，一系列可用编解码方式参考这里 http://fourcc.org/。这里本节代码演示采用 MJPG 格式编码保存。

代码演示可以表示如下：

```cpp
int frame_width = cap.get(CAP_PROP_FRAME_WIDTH);
int frame_height = cap.get(CAP_PROP_FRAME_HEIGHT);

// Define the codec and create VideoWriter object.
// The output is stored in 'outputChaplin.mp4' file.
VideoWriter out("output.mp4",VideoWriter::fourcc('M','J','P',
                'G'),10, Size(frame_width,frame_height));

while(cap.isOpened())
{

    Mat frame;

    // Capture frame-by-frame
    cap >> frame;

    // If the frame is empty, break immediately
    if (frame.empty())
        break;

    // Write the frame into the file 'outputChaplin.mp4'
    out.write(frame);
    imshow("Frame",frame);
    // Wait for 25 ms before moving on to the next frame
    // This will slow down the video
    waitKey(25);
}
// When everything done, release the VideoCapture and
VideoWriter objects
cap.release();
out.release();
```

3.6 回调功能

OpenCV 的 HGUI 模块有几个回调函数，这些回调函数主要实现用户交换响应事件的处理，关于什么是回调，请看下图：



举例说明：当程序在运行中，用户点击了鼠标，就会在系统层面产生一个鼠标事情，应用程序就调用第三方或者底层依赖系统来处理鼠标事件，但是具体需要怎么处理，第三方或者底层依赖系统并不知道，这个时候就需要应用程序实现一个具体处理功能，提供给第三方或者底层应用系统执行，这种处理方式就是回调。OpenCV 中鼠标事件、滚动条事件响应都是通过回调来实现，官方的文档如下：

typedef void(*	**cv::ButtonCallback**) (int state, void *userdata) Callback function for a button created by **cv::createButton**. More...
typedef void(*	**cv::MouseCallback**) (int event, int x, int y, int flags, void *userdata) Callback function for mouse events. see **cv::setMouseCallback**. More...
typedef void(*	**cv::OpenGlDrawCallback**) (void *userdata) Callback function defined to be called every frame. See **cv::setOpenGlDrawCallback**. More...
typedef void(*	**cv::TrackbarCallback**) (int pos, void *userdata) Callback function for Trackbar see **cv::createTrackbar**. More...

3.7 键盘输入响应

OpenCV 中的键盘输入功能主要是通过 waitKey 实现。

```
int cv::waitKey(int delay = 0)
```

其中 delay 表示延时，0 表示一直阻塞，直到用户按下任意一个键程序执行执行下去。下面的代码演示了 e/E 或者 z/Z 被按下显示对应字母，按下 ESC 会退出程序。通过循环读取

运行结果如下：

```
int main(void)
{
    // Open webcam
    VideoCapture cap(0);
    Mat frame;
    int k=0;
    if(!cap.isOpened())
    {
        cout<<"Unable to detect webcam "<<"\n";
        return 0;
    }
    else
    {
        while(1)
        {
            // Capture frame
            cap>>frame;
            if(k==27)
                break;
            // Identify if 'e' or 'E' is pressed
            if(k==101 || k==69)
                putText(frame, "E is pressed, press Z or
                        ESC", Point(100,180),
                        FONT_HERSHEY_SIMPLEX, 1, Scalar(0,
                        255,0), 3);
            // Identify if 'z' or 'Z' is pressed or not
            if(k==90 || k==122)
                putText(frame, "Z is pressed", Point(100,180)
                        , FONT_HERSHEY_SIMPLEX, 1, Scalar(0,
                        255,0), 3);
            imshow("Image",frame);
            // Waitkey is increased so that the display is
            shown
            k= waitKey(10000) & 0xFF;
        }
    }
    cap.release();
    destroyAllWindows();
}
```

执行结果如下：

3.8 使用鼠标

使用鼠标最常见的就是左键与右键点击操作，OpenCV 可以检测窗口内鼠标左键或者右键点击事件，获取鼠标位置信息，实现鼠标事件回调处理。要实现鼠标事件回调，首先调用 namedWindow 创建一个窗口，然后通过 setMouseCallback 在窗口上添加鼠标监听回调功能，具体看本节稍后的代码演示部分。

代码演示部分是通过鼠标在显示图像上绘制一个圆，首先按住鼠标左键确定圆心位置，然后拖动鼠标到你希望的圆半径大小，抬起左键即可绘制，支持多个圆绘制，按字母键 C 实现清除绘制的圆，按 ESC 键退出程序。

鼠标回调实现绘制圆功能的函数

```
// 鼠标回调功能实现
void drawCircle(int action, int x, int y, int flags, void
                *userdata)
{
    // 左键按下
    if( action == EVENT_LBUTTONDOWN )
    {
        center = Point(x,y);
        // Mark the center
        circle(source, center, 1, Scalar(255,255,0), 2,
               CV_AA );
    }
    // 左键抬起
    else if( action == EVENT_LBUTTONUP)
    {
        circumference = Point(x,y);
        // 计算圆半径
        float radius = sqrt(pow(center.x-circumference.x,2)+
                            pow(center.y-circumference.y,2));
        // Draw the circle
        circle(source, center, radius, Scalar(0,255,0), 2,
               CV_AA );
        imshow("Window", source);
    }
}
```

设置鼠标回调支持

```
setMouseCallback("Window", drawCircle);
```

鼠标回调函数说明

```
void cv::setMouseCallback(
    const String &  winname,
    MouseCallback   onMouse,
    void *  userdata = 0
)
```

winname 表示对应窗口名称

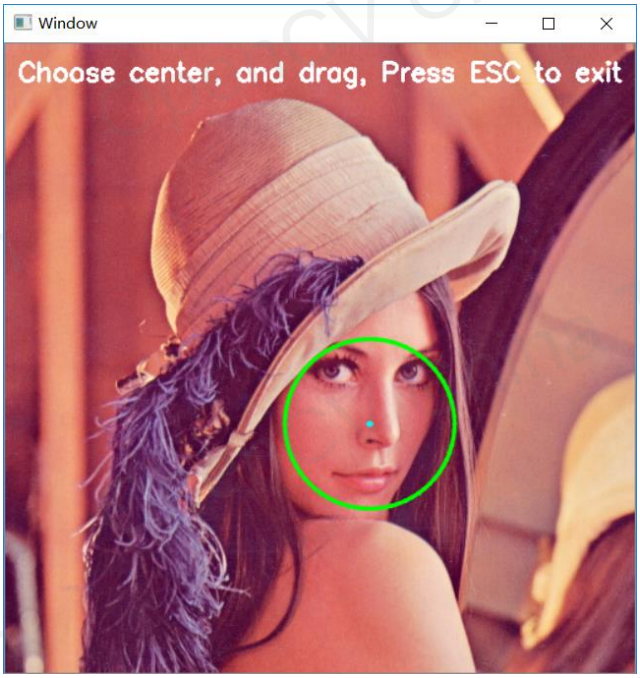
onMouse 回调函数名称

userdata 可选参数，需要回传的数据

主程序运行的代码如下：

```
int main()
{
    source = imread("D:/images/lena.jpg", 1);
    // 清除/恢复时候预留使用对象
    Mat dummy = source.clone();
    namedWindow("Window");
    // 设置鼠标回调
    setMouseCallback("Window", drawCircle);
    int k = 0;
    // 循环监听，直到ESC结束
    while (k != 27)
    {
        imshow("Window", source);
        putText(source, "Choose center, and drag, Press ESC
                to exit and c to clear", Point(10, 30),
                FONT_HERSHEY_SIMPLEX, 0.7, Scalar(255, 255,
                255), 2);
        k = waitKey(20) & 0xFF;
        if (k == 99)
            dummy.copyTo(source);
    }
    return 0;
}
```

运行结果如下：



3.9 使用滚动条作为控制器

本节我们将学习如何在 OpenCV 中使用滚动条。我们将通过阈值操作来演示滚动条(Trackbar)使用。

要使用 Trackbar，首先需要通过 namedWindow 创建一个窗口，然后通过 createTrackbar()功能来创建 Trackbar，在创建时候需要声明窗口名称与回调函数，createTrackbar 函数的解释如下：

```
int cv::createTrackbar(
    const String &  trackbarname,
    const String &  winname,
    int *   value,
    int     count,
    TrackbarCallback    onChange = 0,
    void *  userdata = 0
)
```

Trackbarname 表示创建的 trackbar 名称
winname 表示窗口名称
value 可选的指针值，表示滚动条的滑块当前所在位置，该值会一直跟着滑块位置变化而变化
count 表示滑块移动到最末时对应的最大值，最小值总为 0
onChange 表示回调函数，当拖动滑块时自动调用该函数
userdata 可选的用户数据，可以被传到回调函数中，作为全局变量使用。

下面演示程序实现图像放缩，创建的两个滚动条一个控制放缩尺度，另外一个控制放缩类型(放大/缩小)。创建两个滚动条的代码如下：

```
// 控制放缩的比率
createTrackbar(trackbarValue, windowName, &scaleFactor,
               maxScaleUp, scaleImage);

// 控制放缩类型-放大或者缩小
createTrackbar(trackbarType, windowName, &scaleType, maxType,
               scaleImage);
```

上述代码中
windowName 表示 Trackbar 将会显示的对应窗口名称
scaleFactor 与 scaleType 是拖动 slider 时候会被更新的值
maxScaleUp 与 maxType 是最大值常量，表示 Trackbar 变量最大幅度。
scaleImage 表示回调函数

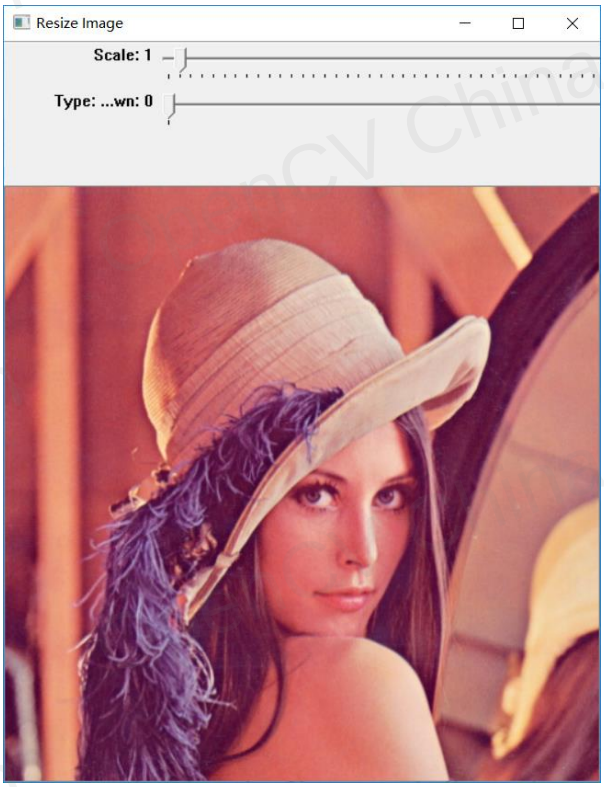
主程序中的代码实现如下：

```
int maxScaleUp = 100;
int scaleFactor = 1;
int scaleType = 0;
int maxType = 1;

string windowName = "Resize Image";
string trackbarValue = "Scale";
string trackbarType = "Type: \n 0: Scale Up \n 1: Scale
                       Down";

void scaleImage(int, void*);

int main()
{
    // 加载图像
    Mat im = imread("D:/images/lena.jpg");

    // 创建窗口
    namedWindow(windowName, WINDOW_AUTOSIZE);

    // 创建Trackbar
    createTrackbar(trackbarValue, windowName, &scaleFactor,
                   maxScaleUp, scaleImage);
    createTrackbar(trackbarType, windowName, &scaleType,
                   maxType, scaleImage);

    scaleImage(25,0);
    // 开始动态监听Trackbar操作
    while (true)
    {
        int c;
        c = waitKey(20);
        if (static_cast<char>(c) == 27)
            break;
    }

    return 0;
}
```

回调函数的代码实现如下：

```
// 回调函数
void scaleImage(int, void*)
{

    // Get the Scale factor from the trackbar
    double scaleFactorDouble = 1 + scaleFactor/100.0;

    if (scaleFactorDouble == 0)
    {
        scaleFactorDouble = 1;
    }

    Mat scaledImage;

    // Resize the image
    resize(im, scaledImage, Size(), scaleFactorDouble,
           scaleFactorDouble, INTER_LINEAR);
    imshow(windowName, scaledImage);
}
```

运行显示结果如下：


Resize Image
Scale: 1
Type: ...wn: 0