《OpenCV3 编程入门》勘误-2015.3

1

正文 P45 页。

'2.2.1 下载安装 CMake

想要在 Windows 平台下生成 OpenCV 的解决方案，需要一个名为 CMake 的开源软件。note：CMake，是“crossplatform make”的缩写，它是一个跨平台的安装（编译）工具，可以用简单的语句来描述所有平台的安装（编译过程）。他能够输出各种各样的 makefile 或者 project 文件，能测试编译器所支持的 C++特性，类似 UNIX 下的 automake。只是 CMake 的组态档取名为 CmakeLists.txt。Cmake 并不直接建构出最终的软件，而是产生标准的建构档（如 Unix 的 Makefile 或 Windows Visual C++的 projects/workspaces），然后再依一般的建构方式使用。这使得熟悉某个集成开发环境（IDE）的开发者可以用标准的方式建构他的软件，这种可以使用各平台的原生建构系统的能力是 CMake 和 SCons 等其他类似系统的区别之处。

修改为：

想要在 Windows 平台下生成 OpenCV 的解决方案，需要一个名为 CMake 的开源软件。

CMake，是“crossplatform make”的缩写，它是一个跨平台的安装（编译）工具，可以用简单的语句来描述所有平台的安装（编译过程）。他能够输出各种各样的 makefile 或者 project 文件，能测试编译器所支持的 C++特性，类似 UNIX 下的 automake。只是 CMake 的组态档取名为 CmakeLists.txt。Cmake 并不直接建构出最终的软件，而是产生标准的建构档（如 Unix 的 Makefile 或 Windows Visual C++的 projects/workspaces），然后再依一般的建构方式使用。这使得熟悉某个集成开发环境（IDE）的开发者可以用标准的方式建构他的软件，这种可以使用各平台的原生建构系统的能力是 CMake 和 SCons 等其他类似系统的区别之处。

2

p350 页，9.2.3 节，下列语句中多了字符串+-**。

程序中新出现的 MatND 类是用于存储直方图的一种数据结构，其用法简单，常常在直方图相关 OpenCV 程序中出现。而+-**程序运行截图如图 9.4、9.5 所示。

修改为：

程序中新出现的 MatND 类是用于存储直方图的一种数据结构，其用法简单，常常在直方图相关 OpenCV 程序中出现。而程序运行截图如图 9.4、图 9.5 所示。

3

P127，5.3.3 节的示例程序，替换成如下代码：

```
//----------------------------【头文件、命名空间包含部分】----------------------
//   描述：包含程序所依赖的头文件和命名空间
//-------------------------------------------------------------------------------
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
using namespace cv;
using namespace std;

//----------------------------------【全局函数声明部分】--------------------------
//   描述：全局函数声明
//-------------------------------------------------------------------------------
bool   MultiChannelBlending();
void ShowHelpText();

//-----------------------------------【main( )函数】--------------------------------
//   描述：控制台应用程序的入口函数，我们的程序从这里开始
//-------------------------------------------------------------------------------
int main(    )
{
    system("color 9F");

    if(MultiChannelBlending( ))
    {
        cout<<endl<<"\n 运行成功，得出了需要的图像~! ";
    }

    waitKey(0);
    return 0;
}

//-----------------------------【MultiChannelBlending( )函数】----------------------
//   描述：多通道混合的实现函数
//-------------------------------------------------------------------------------
```

```
bool  MultiChannelBlending()
{
    //【0】定义相关变量
    Mat srcImage;
    Mat logoImage;
    vector<Mat> channels;
    Mat  imageBlueChannel;

    //=================【蓝色通道部分】=================
    //  描述：多通道混合-蓝色分量部分
    //============================================

    // 【1】读入图片
    logoImage= imread("dota_logo.jpg",0);
    srcImage= imread("dota_jugg.jpg");

    if( !logoImage.data ) { printf("Oh ,no ,读取 logoImage 错误~！\n"); return false; }
    if( !srcImage.data ) { printf("Oh，no，读取 srcImage 错误~！ \n"); return false; }

    //【2】把一个 3 通道图像转换成 3 个单通道图像
    split(srcImage,channels);//分离色彩通道

    //【3】将原图的蓝色通道引用返回给 imageBlueChannel，注意是引用，相当于两者等价，修改其中一个另一个跟着变
    imageBlueChannel= channels.at(0);
    //【4】将原图的蓝色通道的（500,250）坐标处右下方的一块区域和 logo 图进行加权操作，将得到的混合结果存到 imageBlueChannel 中
    addWeighted(imageBlueChannel(Rect(500,250,logoImage.cols,logoImage.rows)),1.0,

    logoImage,0.5,0,imageBlueChannel(Rect(500,250,logoImage.cols,logoImage.rows)));

    //【5】将三个单通道重新合并成一个三通道
    merge(channels,srcImage);

    //【6】显示效果图
    namedWindow(" <1>游戏原画+logo 蓝色通道");
```

```
imshow(" <1>游戏原画+logo 蓝色通道",srcImage);


//==================【绿色通道部分】==================
//    描述：多通道混合-绿色分量部分
//============================================

//【0】定义相关变量
Mat    imageGreenChannel;

//【1】重新读入图片
logoImage= imread("dota_logo.jpg",0);
srcImage= imread("dota_jugg.jpg");

if( !logoImage.data ) { printf("读取 logoImage 错误~！ \n"); return false; }
if( !srcImage.data ) { printf("读取 srcImage 错误~！ \n"); return false; }

//【2】将一个三通道图像转换成三个单通道图像
split(srcImage,channels);//分离色彩通道

//【3】将原图的绿色通道的引用返回给 imageBlueChannel，注意是引用，相当于两者等价，修改其中一个另一个跟着变
imageGreenChannel= channels.at(1);
//【4】将原图的绿色通道的（500,250）坐标处右下方的一块区域和 logo 图进行加权操作，将得到的混合结果存到 imageGreenChannel 中
addWeighted(imageGreenChannel(Rect(500,250,logoImage.cols,logoImage.rows)),1.0,

logoImage,0.5,0.,imageGreenChannel(Rect(500,250,logoImage.cols,logoImage.rows)));

//【5】将三个独立的单通道重新合并成一个三通道
merge(channels,srcImage);

//【6】显示效果图
namedWindow("<2>游戏原画+logo 绿色通道");
imshow("<2>游戏原画+logo 绿色通道",srcImage);
```

```
    //==================【红色通道部分】==================
    //    描述：多通道混合-红色分量部分
    //=============================================

    //【0】定义相关变量
    Mat    imageRedChannel;

    //【1】重新读入图片
    logoImage= imread("dota_logo.jpg",0);
    srcImage= imread("dota_jugg.jpg");

    if( !logoImage.data ) { printf("Oh ,no ,读取 logoImage 错误~！\n"); return false; }
    if( !srcImage.data ) { printf("Oh，no，读取 srcImage 错误~！  \n"); return false; }

    //【2】将一个三通道图像转换成三个单通道图像
    split(srcImage,channels);//分离色彩通道

    //【3】将原图的红色通道引用返回给 imageBlueChannel，注意是引用，相当于两者等价，修改其中一个另一个跟着变
    imageRedChannel= channels.at(2);
    //【4】将原图的红色通道的（500,250）坐标处右下方的一块区域和 logo 图进行加权操作，将得到的混合结果存到 imageRedChannel 中
    addWeighted(imageRedChannel(Rect(500,250,logoImage.cols,logoImage.rows)),1.0,

    logoImage,0.5,0.,imageRedChannel(Rect(500,250,logoImage.cols,logoImage.rows)));

    //【5】将三个独立的单通道重新合并成一个三通道
    merge(channels,srcImage);

    //【6】显示效果图
    namedWindow("<3>游戏原画+logo 红色通道  ");
    imshow("<3>游戏原画+logo 红色通道  ",srcImage);

    return true;
}
```

4

封底。读者热评“讲解得挺全面的。原理、实现、具体代码，非常适合初学者实用的教程——gmf_henu”

改为“讲解得挺全面的。原理、实现、具体代码，非常适合初学者使用的教程——gmf_henu”，或“讲解得挺全面的。原理、实现、具体代码，非常适合初学者的实用的教程——gmf_henu”