**OpenCV3.2.0 + CUDA8.0 + Visual Studio 2013 x64 配置**

## 2017.5.18

OpenCV 官网提供的预编译库并不包含完整的 CUDA 功能，所以要想体验 GPU 加速技术带给 OpenCV 的变化，只能自己重新编译整个库。

1. **下载安装 CUDA Toolkit**

[https://developer.nvidia.com/](https://developer.nvidia.com/cuda-downloads)下载最新的 CUDA Tookit 。验证 CUDA 是否安装正确，可以尝试编译任意一个 CUDA 自带的样例程序，图 1。

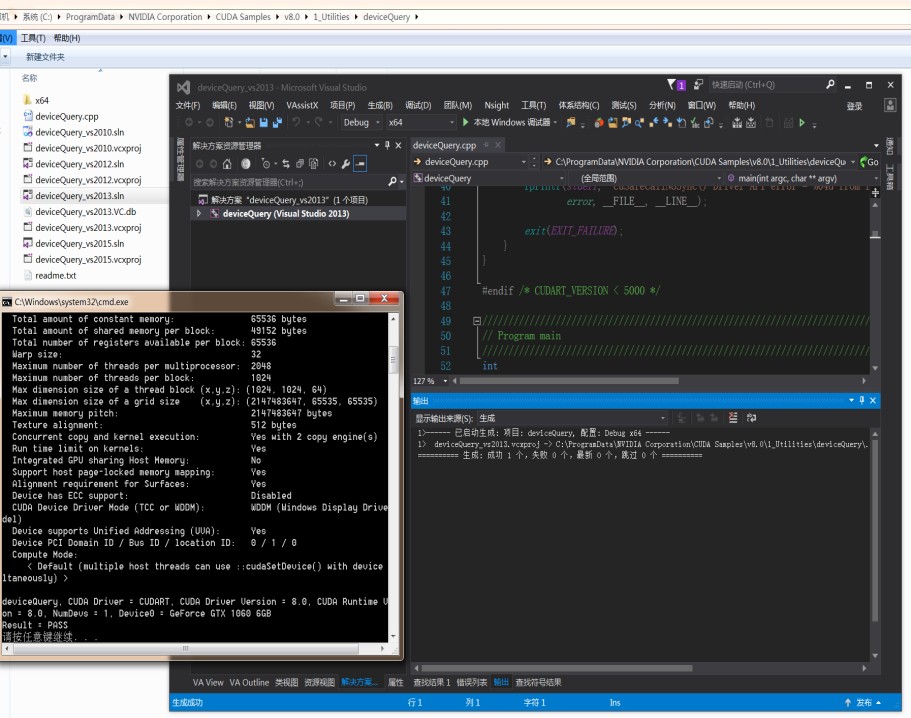
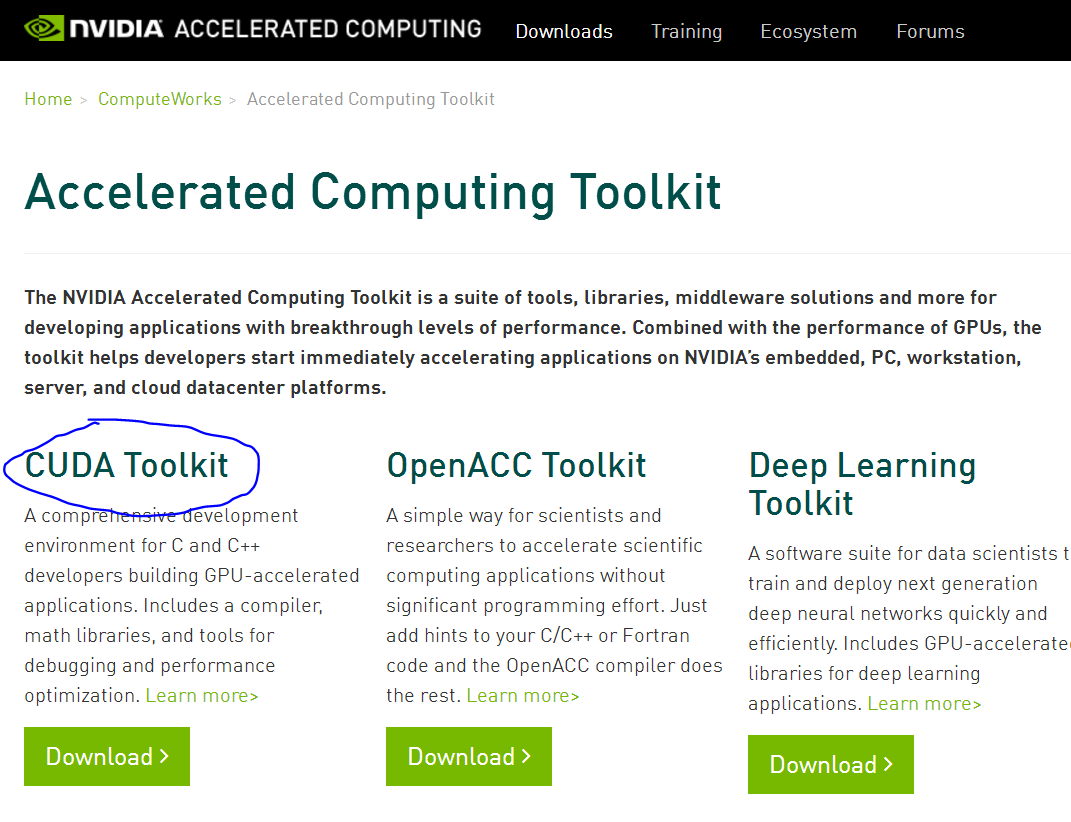


图 1

注意：CUDA8.0（支持 Pascal 系列 GPU）可以兼容 opencv3.2 和 opencv2.4.13，2016 年之前发布的 opencv 版本存在部分不兼容。如果显卡是 GTX1060 或以上，最好采用 CUDA8.0+ opencv3.2/ opencv2.4.13+VS(13/15) ×64 的搭配。

1. **下载安装 OpenCV 和 CMake**

<http://opencv.org/> <http://www.cmake.org/>

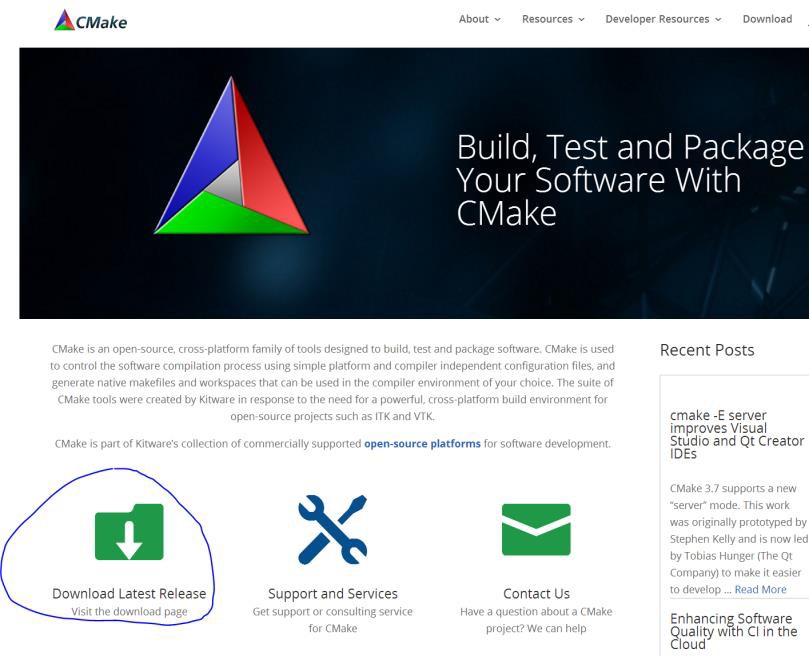
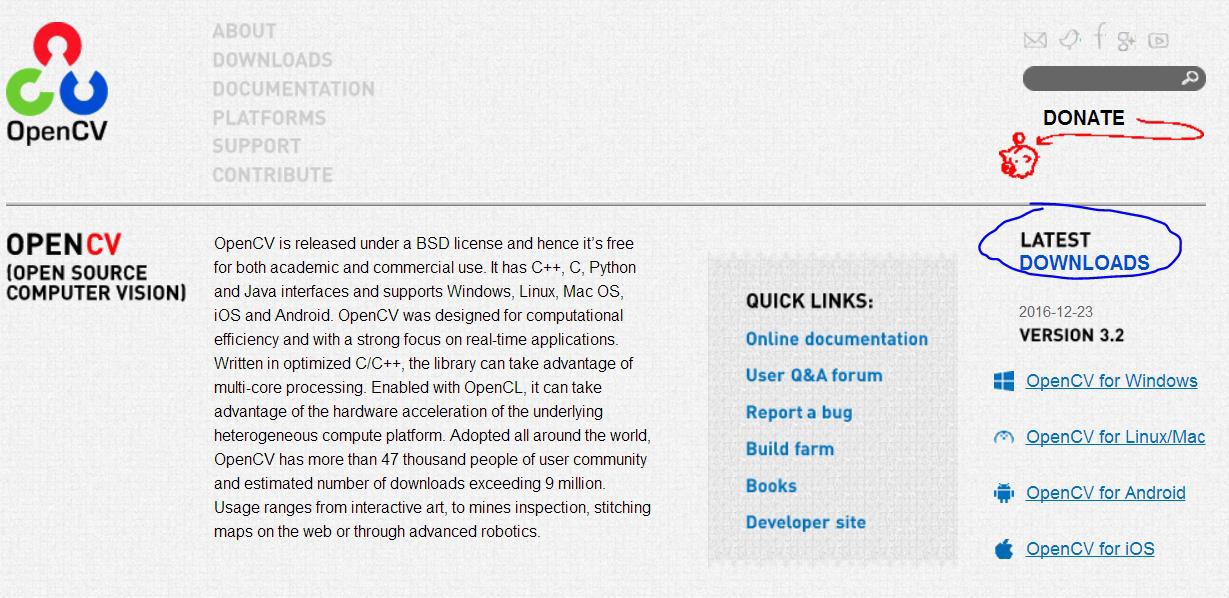


图 2

1. **CMake 配置**

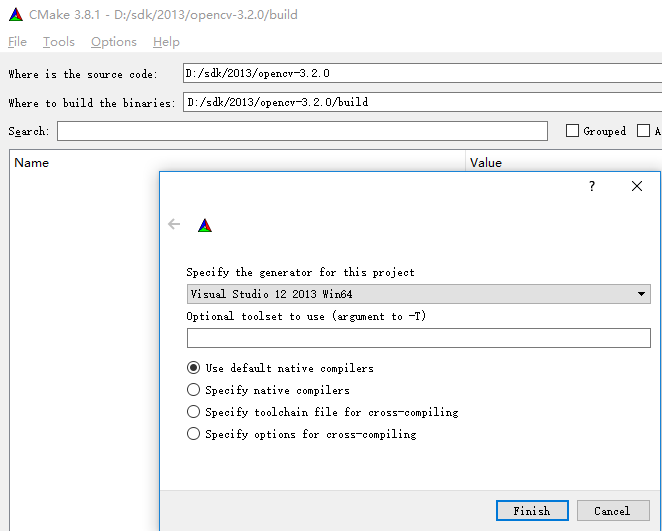
****

图 3

在 CUDA Toolkit 8.0(7.5 也一样)中，虽然给出了适用 Win32 和 x64 两种目标架构的库，但像 cufft、npps、nvblas 等 OpenCV 所需的库只有 x64 版本。这也限制了我们只能编译 64 位的 OpenCV 库，而且在今后的编程中也要编写针对 x64 架构的程序。

1. **CMake 中的选项**

(1)

**不要勾选 BUILD\_CUDA\_STUBS 。我们需要勾选的是：**

**WITH\_CUDA、WITH\_CUFFT、WITH\_EIGEN 、WITH\_IPP、WITH\_CUBLAS、 WITH\_OPENGL、WITH\_OPENMP、WITH\_TBB、WITH\_NVCUVID。**

其中 **WITH\_CUDA、WITH\_CUFFT、WITH\_EIGEN、WITH\_IPP** 是默认开启的。（未使用 TBB 可以不开启 WITH\_TBB） 。

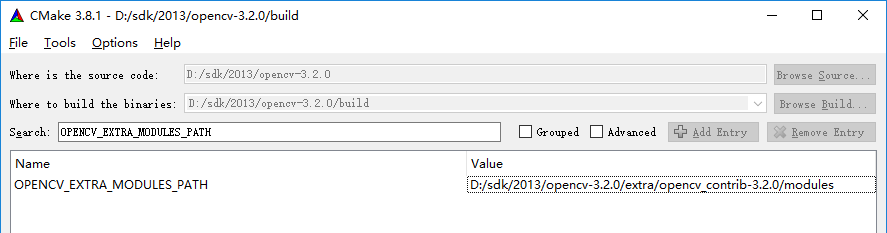
(2)

**CUDA\_FAST\_MATH 不建议勾选**，是 CUDA 的快速数学库，牺牲精度谋求速度，如果只要求单精度浮点可以选择该项。

**BUILD\_opencv\_world不建议勾选**，最后生成的库会合并为两项 **opencv\_world320.dll** 以及 **opencv\_ffmpeg320.dll**，这样做的好处是只需要引用两个库，缺点是很多可能用不着的库在运行时会加载。比如**编译好后的 opencv\_world320d.dll 有 900 多 MB。**

**OPENCV\_ENABLE\_NONFREE 勾选（包含 SURF 等受到专利保护的算法），OPENCV\_EXTRA\_MODULES\_PATH 需要填写正确的额外库路径。**

**（文件下载地址** [**https://github.com/opencv/opencv\_contrib**](https://github.com/opencv/opencv_contrib)**），下面的图需要新建extra文件夹**

****

**图4**

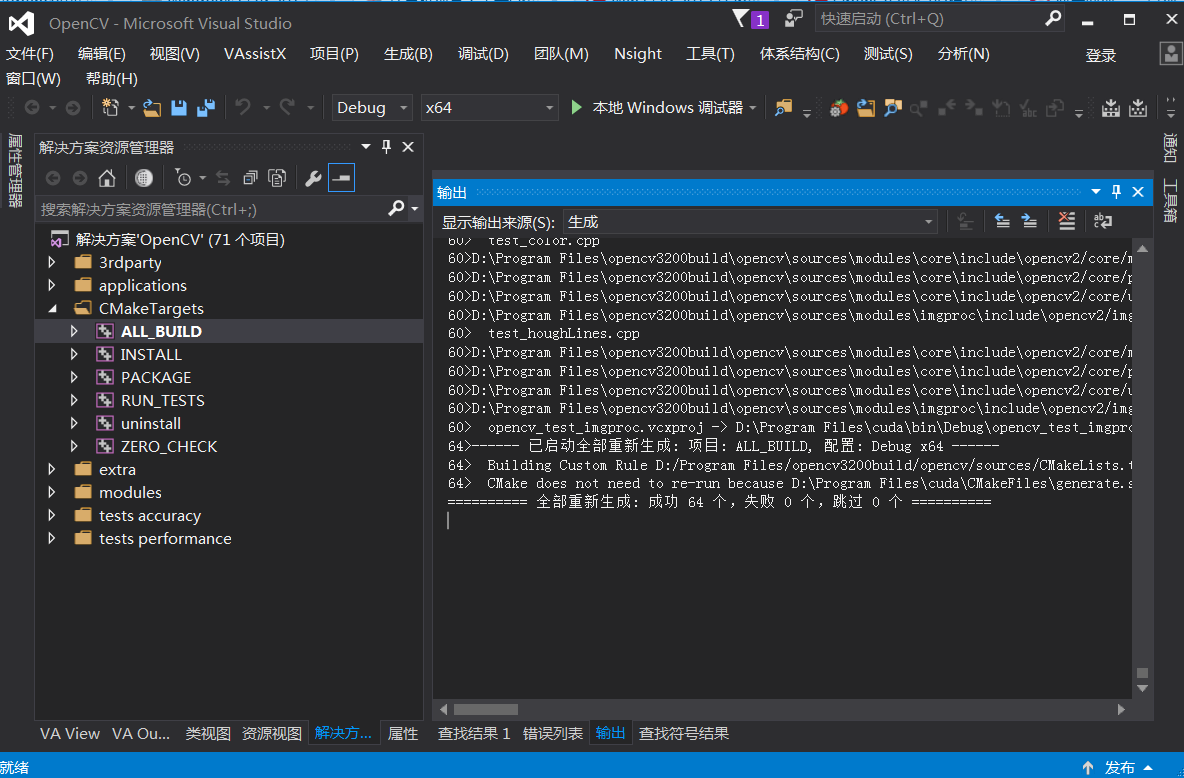
依次点击 Configure、Generate 。

Configure 的过程中可能会下载需要的文件，比如 ippicv\_windows\_20151201.zip 等，也可以从其他地方下载好后复制到相关位置。

1. **用 Visual Studio 编译 OpenCV**

Generate done 完成后，点击 Open Project。首先在 Debug 模式和 Release 模式下，分别 Build **ZERO\_CHECK** 项目，看两次编译是否产生错误。如果两次编译成功，则进行下一步：

**依次进行 ALL\_BUILD（生成很多 OpenCV 项目文件，D+R** 耗时 6 Hours**）、INSTALL（在 build 下生成 install 文件夹，**耗时 5 mins**） 。**



1. **配置属性表**

参考props文件

## 测试

#include "stdafx.h"

#include <iostream>

#include "opencv2/core/core.hpp"

#include <opencv2/highgui/highgui.hpp>

#include "opencv2/core/cuda.hpp"

#include <opencv2/cudaimgproc.hpp>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 需要注意的是，在所有使用GPU模块的函数之前，最好需要调用函数gpu::getCudaEnabledDeviceCount，

如果这个函数返回值为0，同时你在命令行中能够看到“CUDA is no support”的错误， 说明没有编译成功

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

using namespace *cv*; using namespace *std*;

int *main*()

{

int num\_devices = *cuda*::*getCudaEnabledDeviceCount*();

if (num\_devices <= 0)

{

*cout* << "There is no device." << *endl*; return -1;

}

int enable\_device\_id = -1;

for (int i = 0; i < num\_devices; i++)

{

*cuda*::*DeviceInfo* dev\_info(i); if (dev\_info.*isCompatible*())

{

enable\_device\_id = i;

}

}

if (enable\_device\_id < 0)

{

*cout* << "GPU module isn't built for GPU" << *endl*; return -1;

}

*cuda*::*setDevice*(enable\_device\_id); // 设置当前使用的CUDA

*cout* << "GPU is ready, device ID is " << num\_devices << "\n";

*Mat* src\_image = *imread*("boldt.jpg", 1);

*Mat* dst\_image;

*cuda*::*GpuMat* d\_src\_img(src\_image);

*cuda*::*GpuMat* d\_dst\_img;

*cuda*::*cvtColor*(d\_src\_img, d\_dst\_img, *COLOR\_BGR2GRAY*); d\_dst\_img.*download*(dst\_image);

*namedWindow*("test", 0); *imshow*("test", dst\_image); *waitKey*(0);

return 0;

}

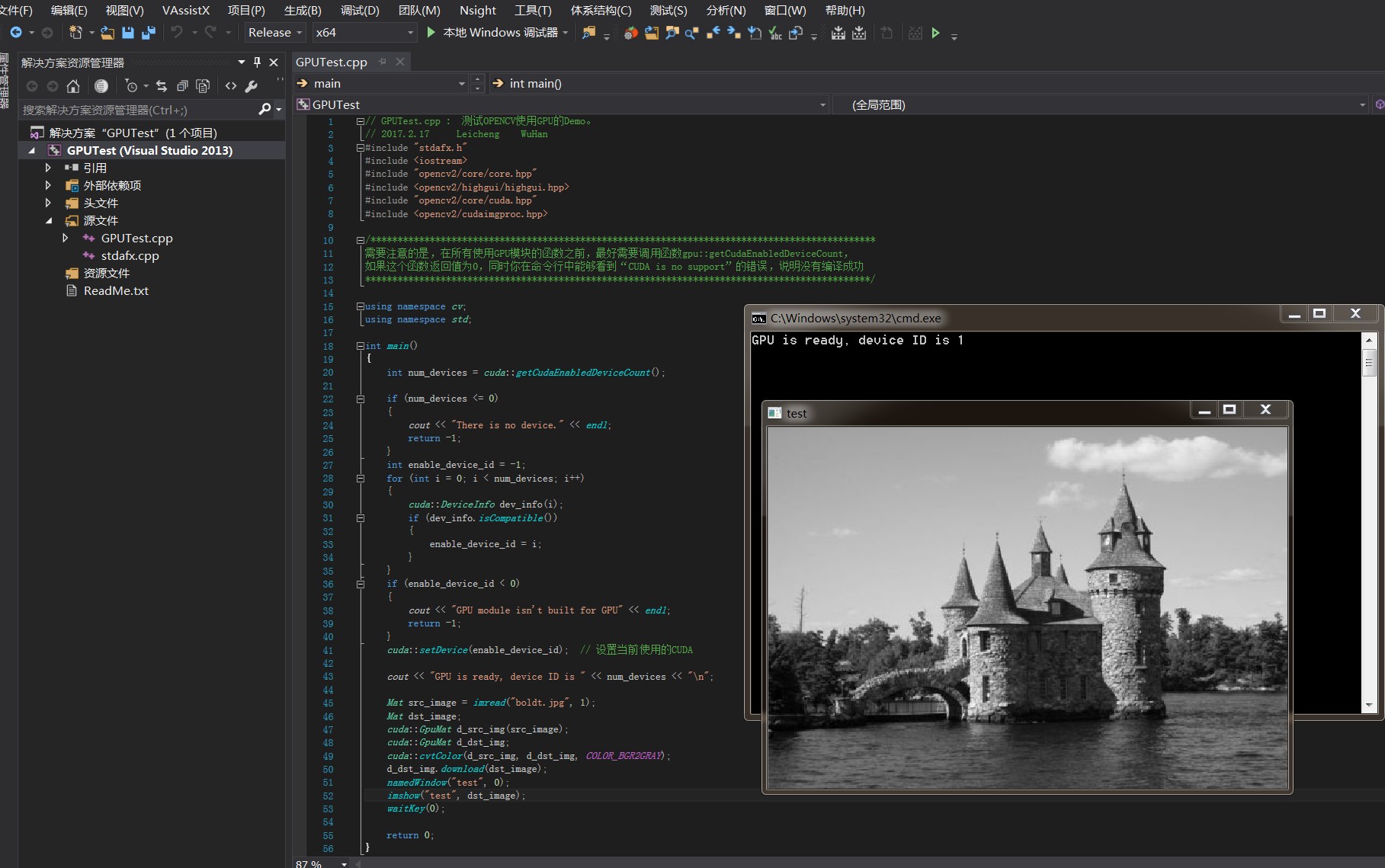


图 8