Visual Studio 2015 + OpenCV3.4.0 + OpenCV\_contrib3.4.0+CUDA8.0+TBB

主要参考：<https://blog.csdn.net/luqiang6q/article/details/78887670>

1. 背景

测试Windows环境下VS中OpenCV利用CUDA进行加速的效果如何，需要进行相关的环境配置以及测试。

配置中所需要的一些文件以及最后得到的库一并在附件中，请在使用前再三确认是否是相同的配置需求。

例程中使用的为 Win 10系统+Visual Studio 2015 update 3+CMake 3.12.0 + OpenCV 3.4.0 +OpenCV\_contrib 3.4.0 + CUDA 8.0.61 + TBB 2018\_20170919 + GTX 960M（Maxwell架构）

1. 预先准备

在进行配置前需要下载或者安装如下程序：

Visual Studio：由于OpenCV 3.1.0以后的版本只支持2015(vs14)以上，故在此以2015版本为例。微软官网可以下载，旧版本的下载可以百度得到。

OpenCV：由于OpenCV 3.1.0及之前版本在使用CUDA时会报错，因而推荐只用3.2.0或者2.4.13及以上版本，在此使用的是3.4.0。下载地址：

<https://opencv.org/releases.html> （需添加环境变量）

OpenCV\_contrib: 3.X版本后一些附带的实验性的功能均放置于contrib扩展包，下载相匹配的版本即可，这里使用的是OpenCV\_contrib 3.4.0。下载地址：<https://github.com/opencv/opencv_contrib/releases>

CUDA: NVIDIA的GPU加速工具，8.0版本相对稳定，这里使用的是8.0.61。多说一句，安装CUDA时似乎是需要Visual Studio 2015 (v140) 平台工具集，使用2017和2013版本似乎都会报错。下载地址：

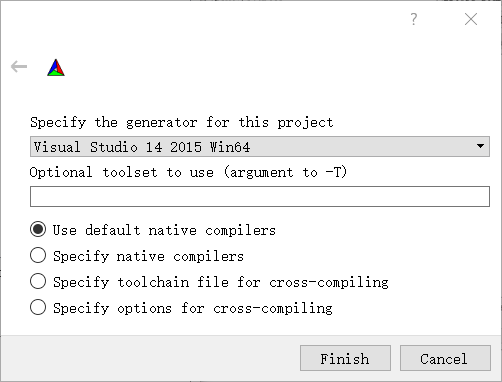
<https://developer.nvidia.com/cuda-80-ga2-download-archive> （需添加环境变量）

TBB： intel threading building blocks (tbb)，Intel对于并行计算的优化库。下载地址：<https://software.intel.com/en-us/intel-tbb> （需添加环境变量）

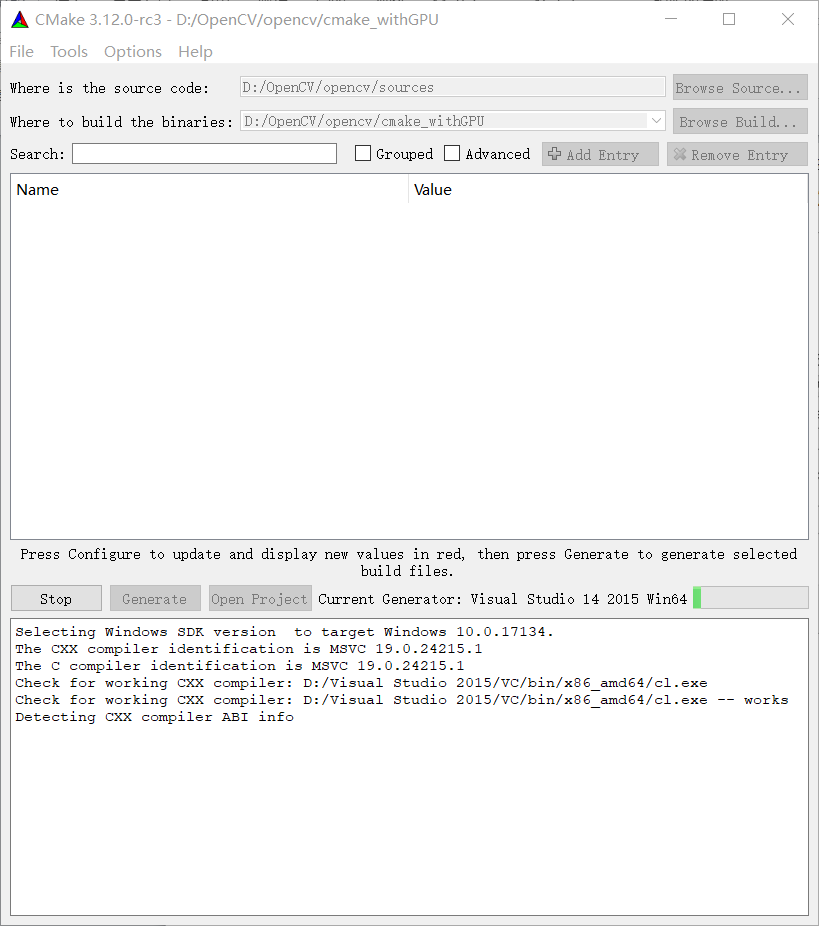
1. 配置流程
   1. 安装完成2中所述的各个软件
   2. 打开Cmake进行第一次编译



编译过程一般需要3-4次不等。首先在CMake中设置好opencv源代码(source code)，目标地址，然后选择Configure选项，选择合适的编译器（Visual Studio 2015/vs14），然后单击finish进行第一次编译。

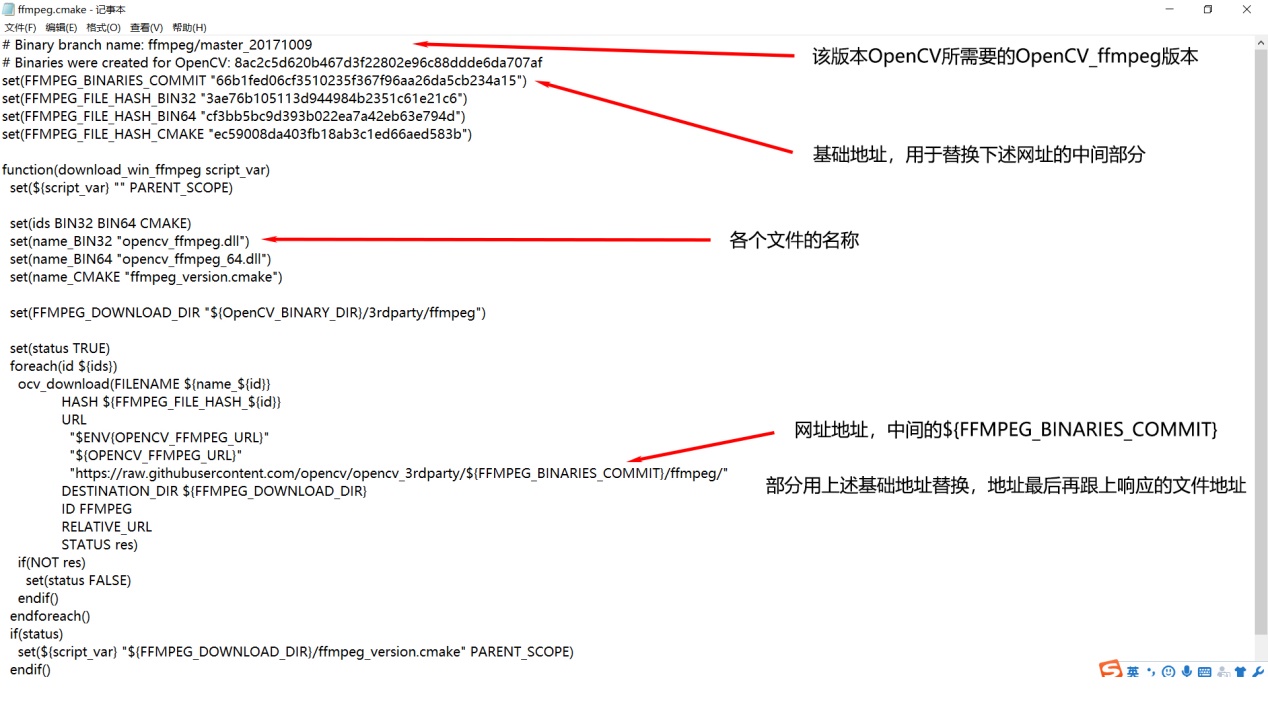


编译过程中出现下述情况为正常，如果出错，原因一般为本地编译器选择错误或者CMake装在中文路径下。



一般编译过程中会出现需要下载文件的情况。通常来说，需要下载的文件包括有：ffmpeg\_version.cmake、opencv\_ffmpeg.dll、opencv\_ffmpeg\_64.dll以及face\_landmark\_model.dat等多个文件。正常来说下载应该是自动进行不会出现问题，但是在不翻墙的前提下经常会出现 “XXXXX **Time-out** XXXXXX”的超时错误提示。所以可进行手动下载并放入响应的文件夹中。本版本所需要的这几个文件中上述四个较大，比较容易超时的文件已经一并附上。需要注意的是，由于各个版本的不同，使用不同版本的opencv所需要的文件是不同的，贸然进行替换会导致后续编译出错。具体各个版本的编译所需要的编译结果在各文件原本应处位置下的CMake文件中会有所提示。

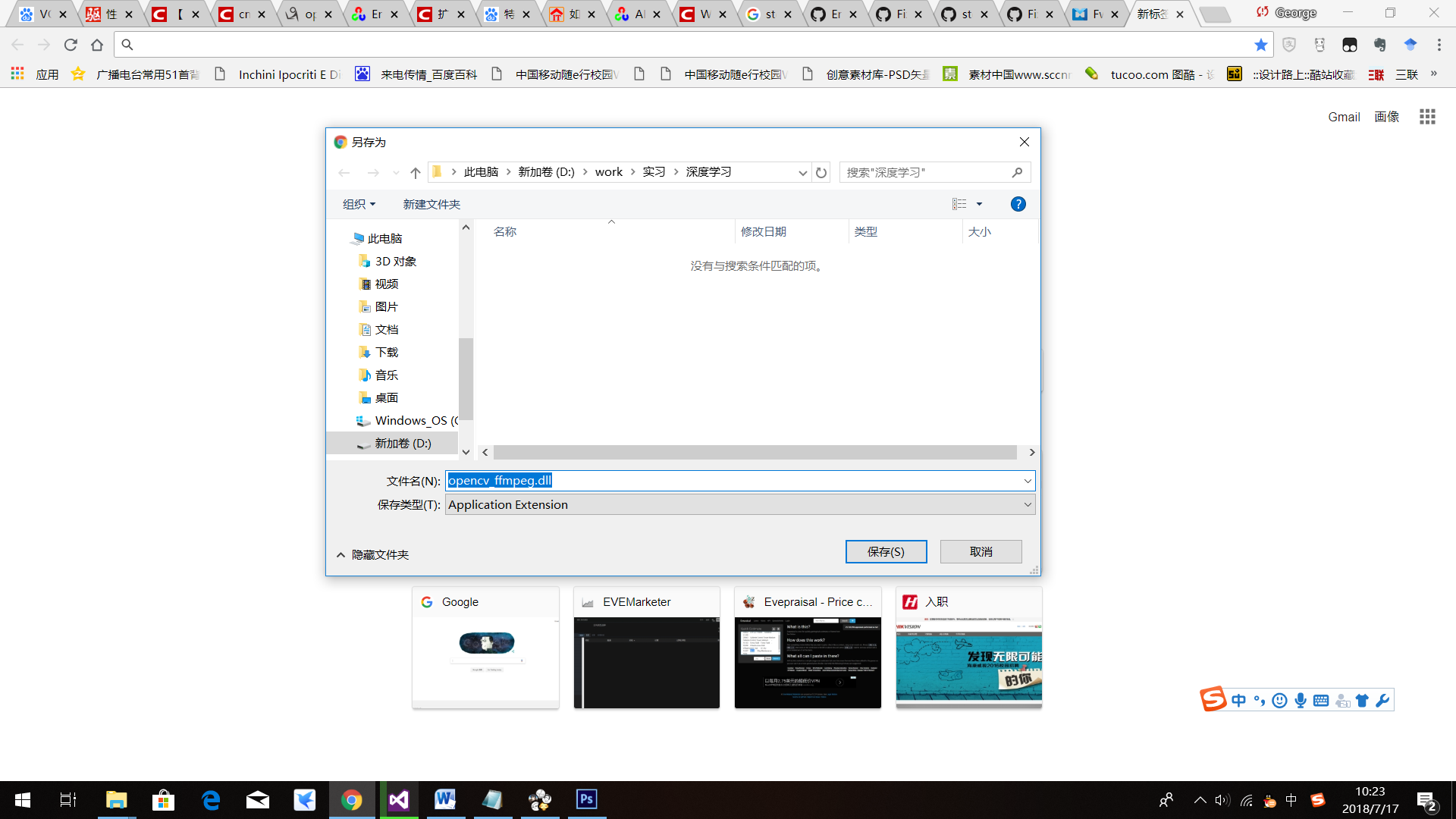
关于ffmpeg的三个文件应位于.\opencv\source\3rdparty\ffmpeg文件夹下，相同版本的可以直接放入其中。不同版本的则应查看CMake文件如下：



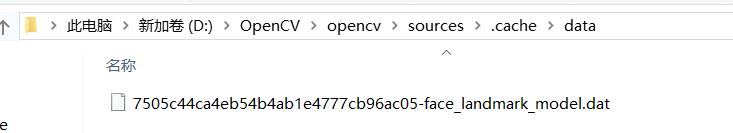
例如要下载opencv\_ffmpeg.dll，则相应网址为：

C:\Users\lenovo\AppData\Roaming\Tencent\Users\564810049\QQ\WinTemp\RichOle\51R`BP%`SY9)~NC@}14@LIH.png

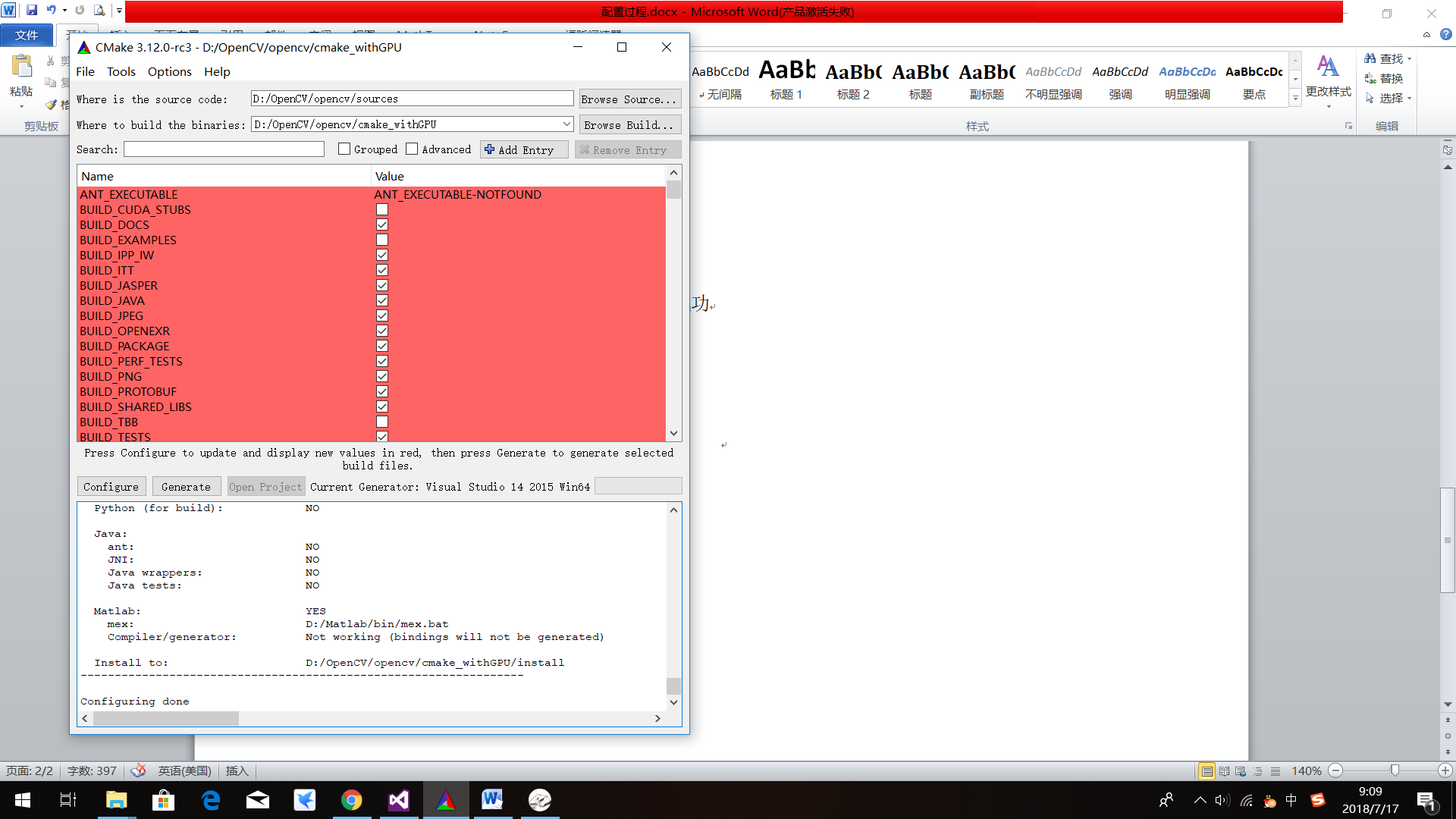
回车会自动弹出对话框进行下载：



face\_landmark\_model.dat文件应位于 .\opencv\sources\.cache\data文件夹下，以相同的方法进行下载，需要注意的是该文件的CMake文件写的比较乱，看起来会有点复杂。另外该文件需要在文件名之前加上其HASH值：



完成上述操作后，出现如下图这个状态则表明第一次编译成功。实际操作中建议先进行一次编译，如若无法下载则再进行手动下载。



* 1. 第二次编译

第二次编译主要添加OpenCV\_contrib库以及CUDA的支持，同时进行一些设置。

找到OPENCV\_EXTRRA\_MODULES\_PATH项，后面添加opencv\_contrib中的modules目录。注意！！选择路径时请使用右边的选择按钮而不是手动输入，windows资源管理器中自带的为右斜杠，而Cmake要求的为左斜杠，会出现奇怪的报错。

勾选选中WITH\_CUDA，WITH\_FFT，WITH\_CUBLAS，WITH\_TBB，WITH\_OPENGL选项。

取消勾选Build\_tests和Build\_perf\_tests选项。

这是OpenCV 3.X版本开始至今的一个bug，与CUDA中似乎会有冲突。该报错只在开启WITH\_OPENCL选项时出现。若不取消勾选会导致后续在VS编译中出现如下报错：

Severity Code Description Project File Line Error C2264 'std::tuple<cv::size,perf::`anonymous-namespace'::matdepth>::tuple': error in function definition or declaration; function not called opencv\_perf\_core

~~Github中提到了另一种解决办法是对三个.cpp文件修改源代码：~~[~~https://github.com/opencv/opencv/issues/6716~~](https://github.com/opencv/opencv/issues/6716)使用后还是出现上述问题。暂时还是推荐取消勾选

若安装有Matlab取消勾选WITH\_MATLAB选项，之后无需用到。

若安装Python、Java也应考虑取消勾选相关选项，尽量避免同时编译多个平台。

在CUDA\_GENERATION下选择合适的显卡架构，具体架构可参见NVIDIA官网。（这里由于便于使用并没有选择这一项，应该问题也不大）

（一般来说16年及以前的显卡可能是Maxwell架构，GTX 10X系列为Pascal架构，最新的Tesla V100为Volta架构）

然后继续configure

* 1. 第三次编译

第三次编译加入TBB库支持（仍然注意上述路径问题）

TBB\_ENV\_iNCLUDE: 添加tbb的include的目录

（如C:\Users\6Qiang\Downloads\tbb2018\_20171205oss\include）

TBB\_ENV\_LIB: 添加tbb的tbb.lib文件

（如C:\Users\6Qiang\Downloads\tbb2018\_20171205oss\lib\intel64\vc14\tbb.lib）

TBB\_ENV\_LIB\_DEBUG: 添加tbb的tbb\_debug.lib文件

（如:\Users\6Qiang\Downloads\tbb2018\_20171205oss\lib\intel64\vc14\tbb\_debug.lib）

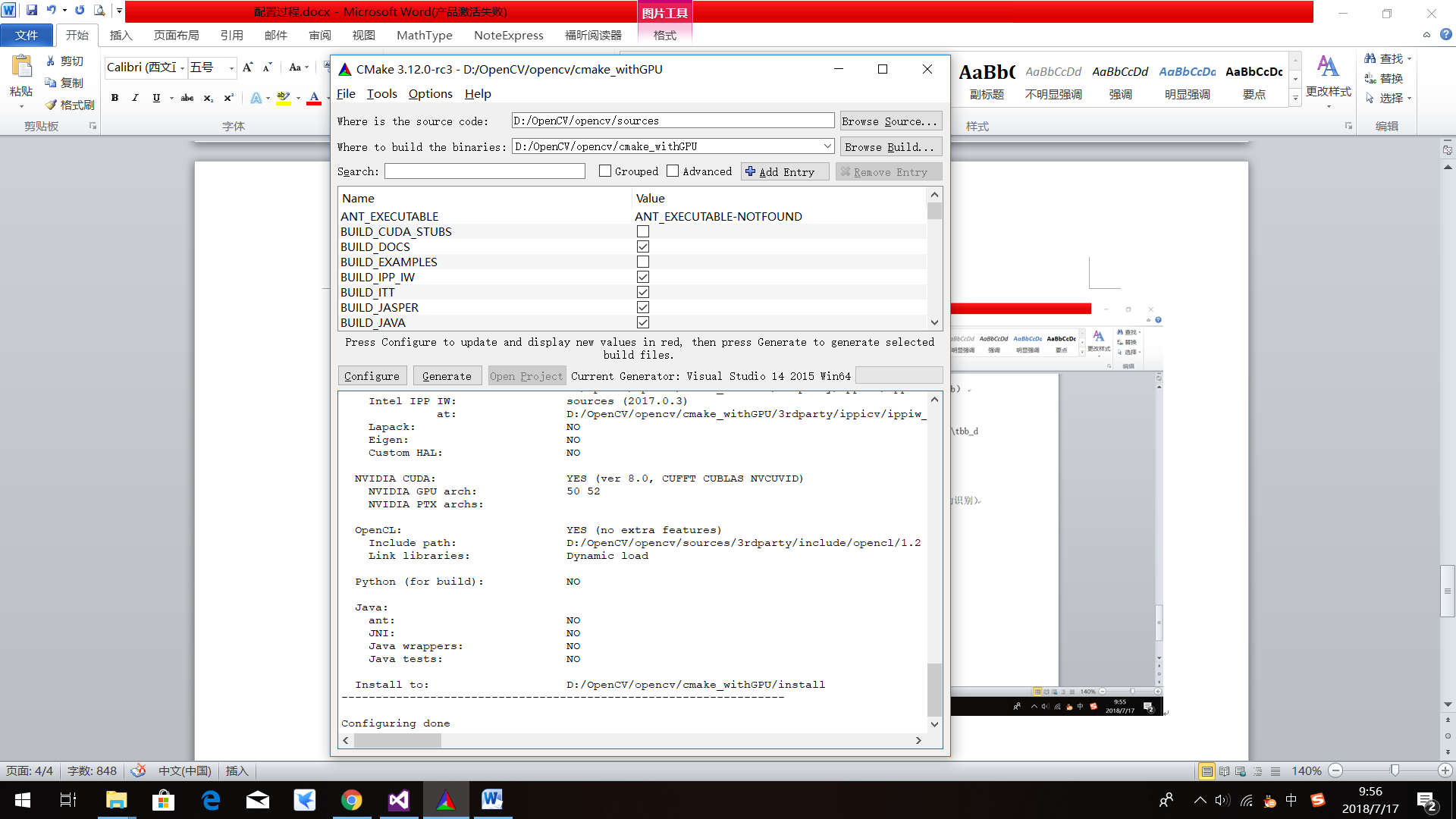
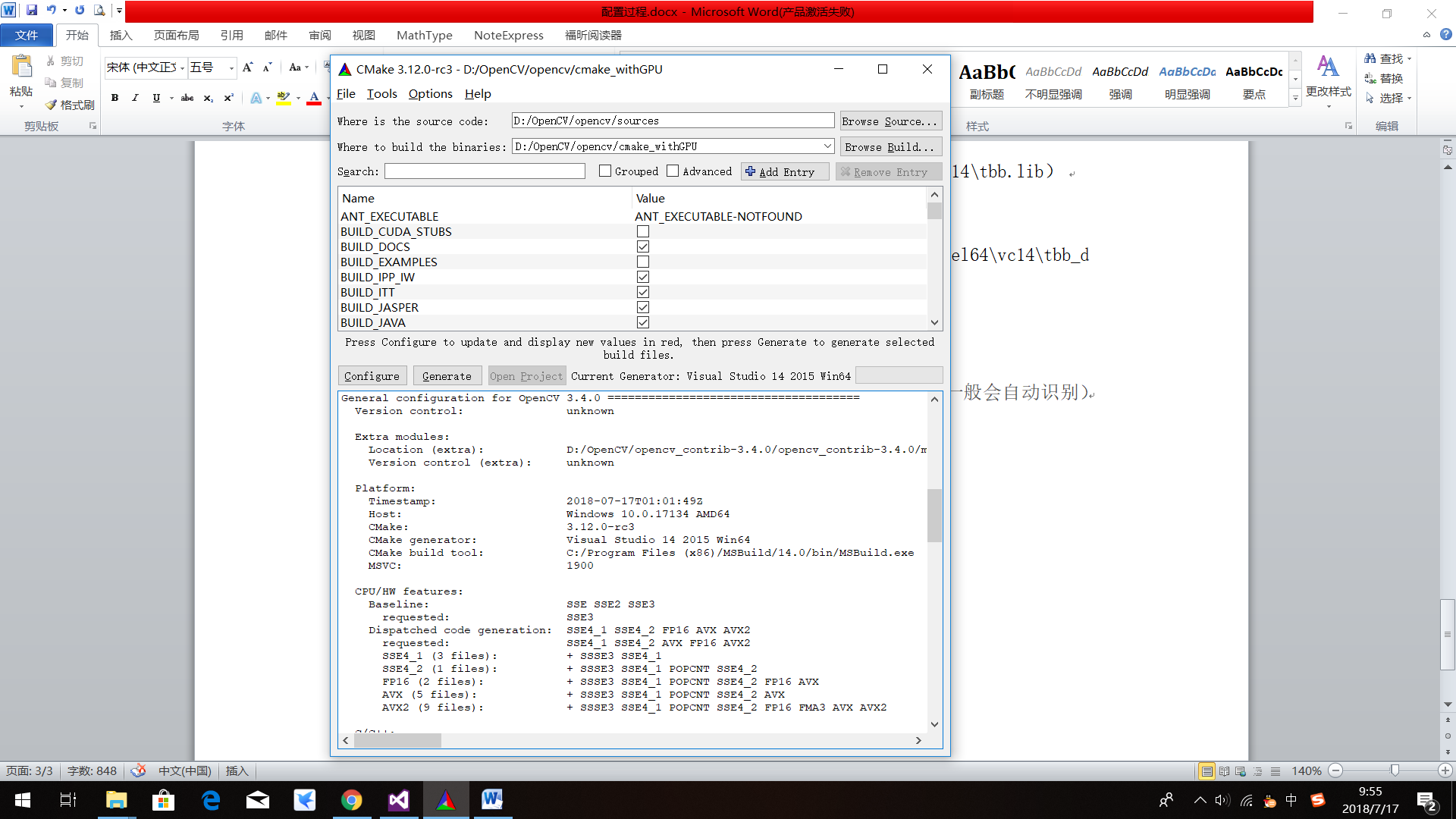
然后继续configure

* 1. 第四次编译

第四次编译添加TBB\_VER\_FILE：添加tbb\_stddef.h的目录 （一般会自动识别）

然后继续configure

* 1. 结束编译之后一般会如下图所示

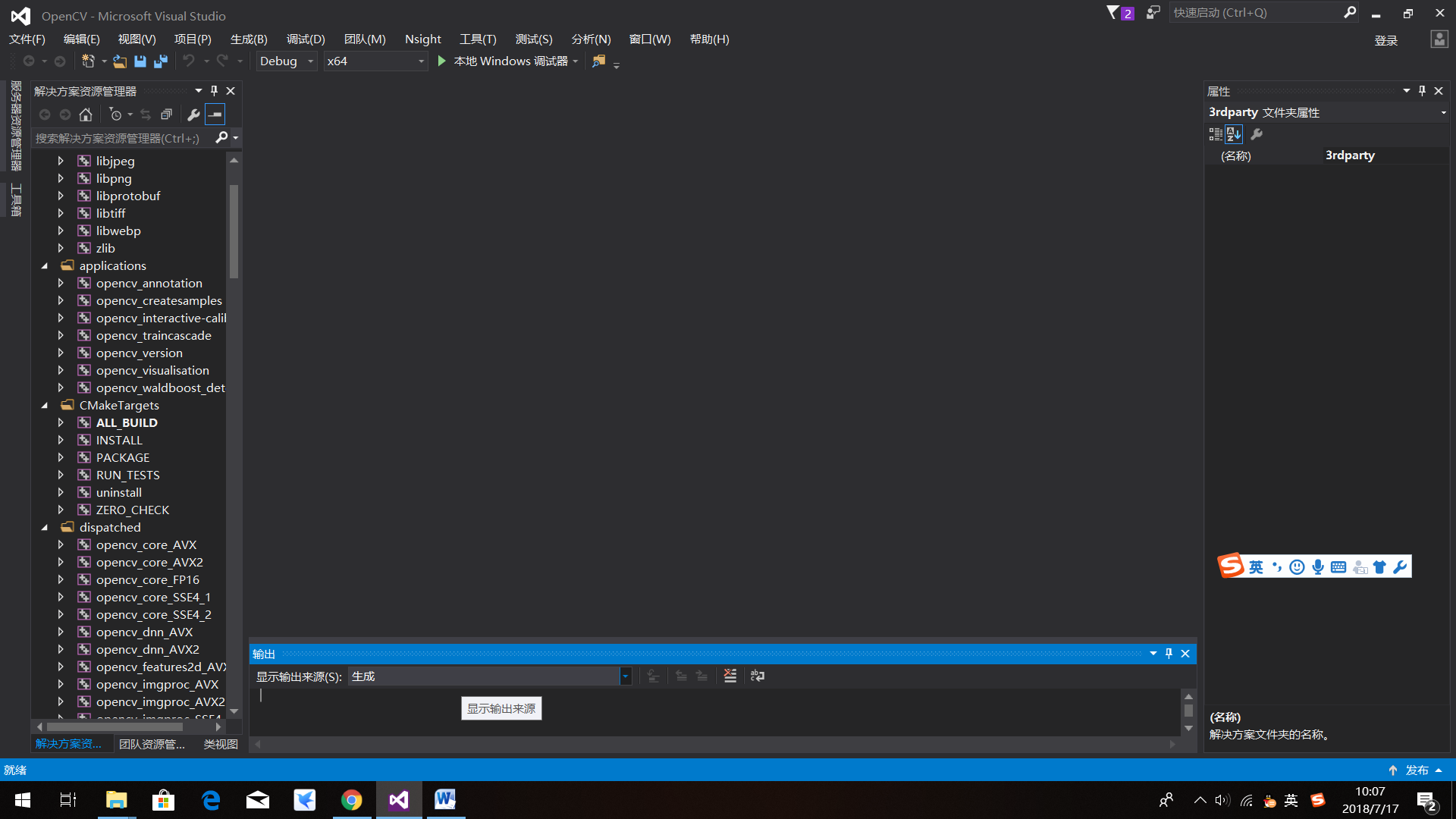


按上述步骤之后，若出现configuring done则表示CMake编译完成。需要着重检查其中几项内容：CMake generator, OpenGL support, Video I/O(FFMPEG), Intel IPP, NVIDIA CUDA, OpenCL, Parallel framework。

若上述选项都检查无误，则可单击Generate生成CPP文件。若提示Generating done则表示生成成功。关闭Cmake。

* 1. Visual Studio 编译

打开Visual Studio平台，在刚才保存的目标文件夹下找到OpenCV.sln并打开。



分别在debug模式下对左边CMakeTargets文件夹下的ALL\_BUILD进行生成，然后对INSTALL右键选择仅用于项目-仅生成。在Release模式下重复上述操作，最终可以在目标路径的Install文件夹下得到编译完成后的库文件，可以直接导出使用。在VS中整个编译过程比较慢，在debug和release模式中分别需要2个小时以上的时间。

**从这里之前的配置过程可以直接使用别人已经配置好的install文件下的文件使用，无需再重新开始配置。需要注意的是要保证OpenCV的版本支持CUDA和相应的Visual Studio，（似乎还需要是相同的显卡）。**

* 1. 编译完成后附加依赖库

在用户变量中添加：（变量名：变量值）

opencv\_cuda：.\cmake\_withGPU\install

在系统变量Path中添加：

.\cmake\_withGPU\install\x64\vc14\bin

有说法是装过opencv的需要在系统变量中将此路径至于opencv本身的路径前，不然可能会报错。未经实验，不过我乖乖照做了。

重启计算机。不重启会报错

Debug模式下：

新建项目，在包含目录中添加:

.\cmake\_withGPU\install\include

C:\Program Files\NVIDIA GPU ComputingToolkit\CUDA\v8.0\include（视CUDA位置）

在库目录中添加

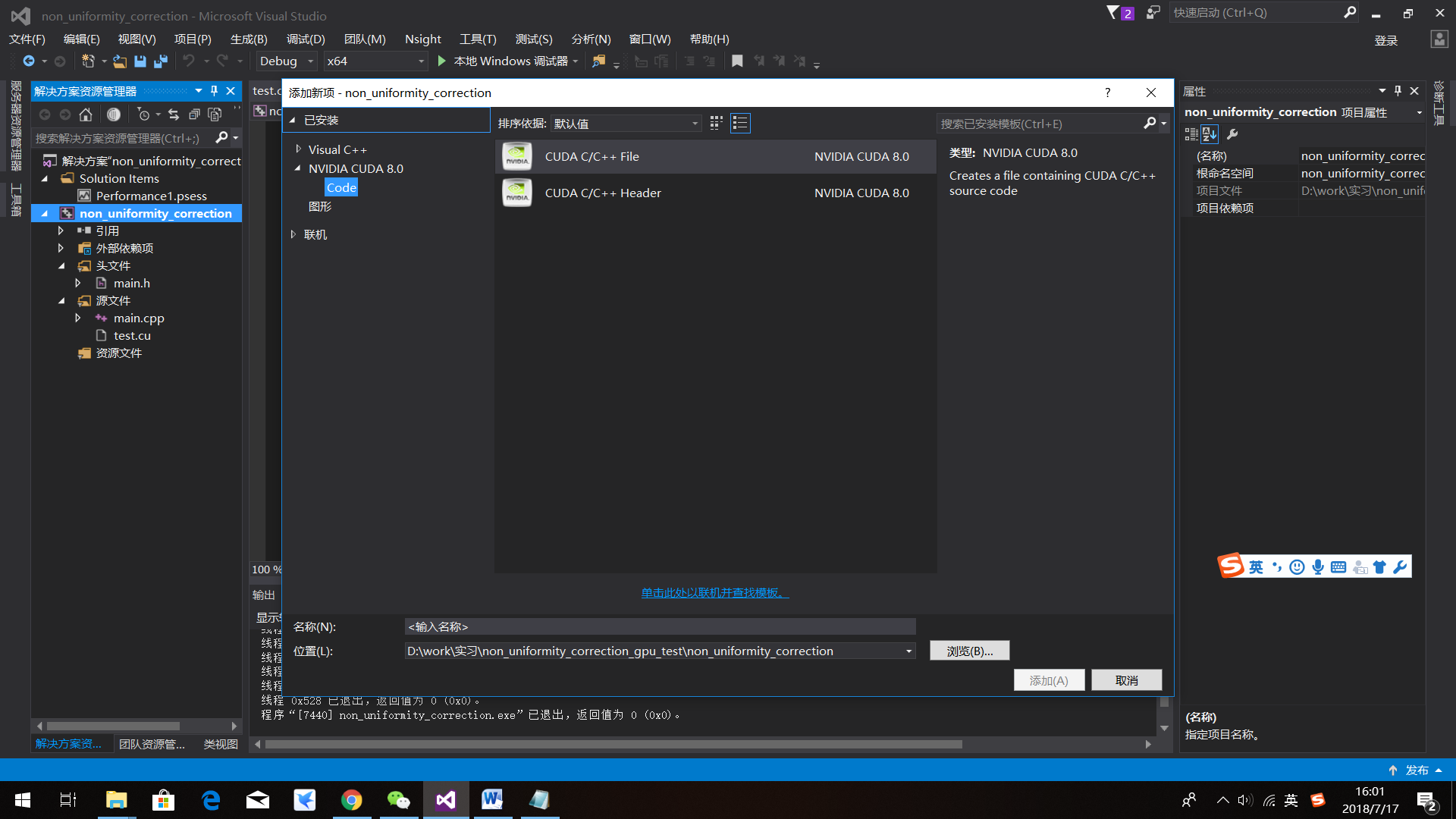
.\cmake\_withGPU \install\x64\vc14\lib

C:\Program Files\NVIDIA GPU ComputingToolkit\CUDA\v8.0\lib\x64（视CUDA位置）

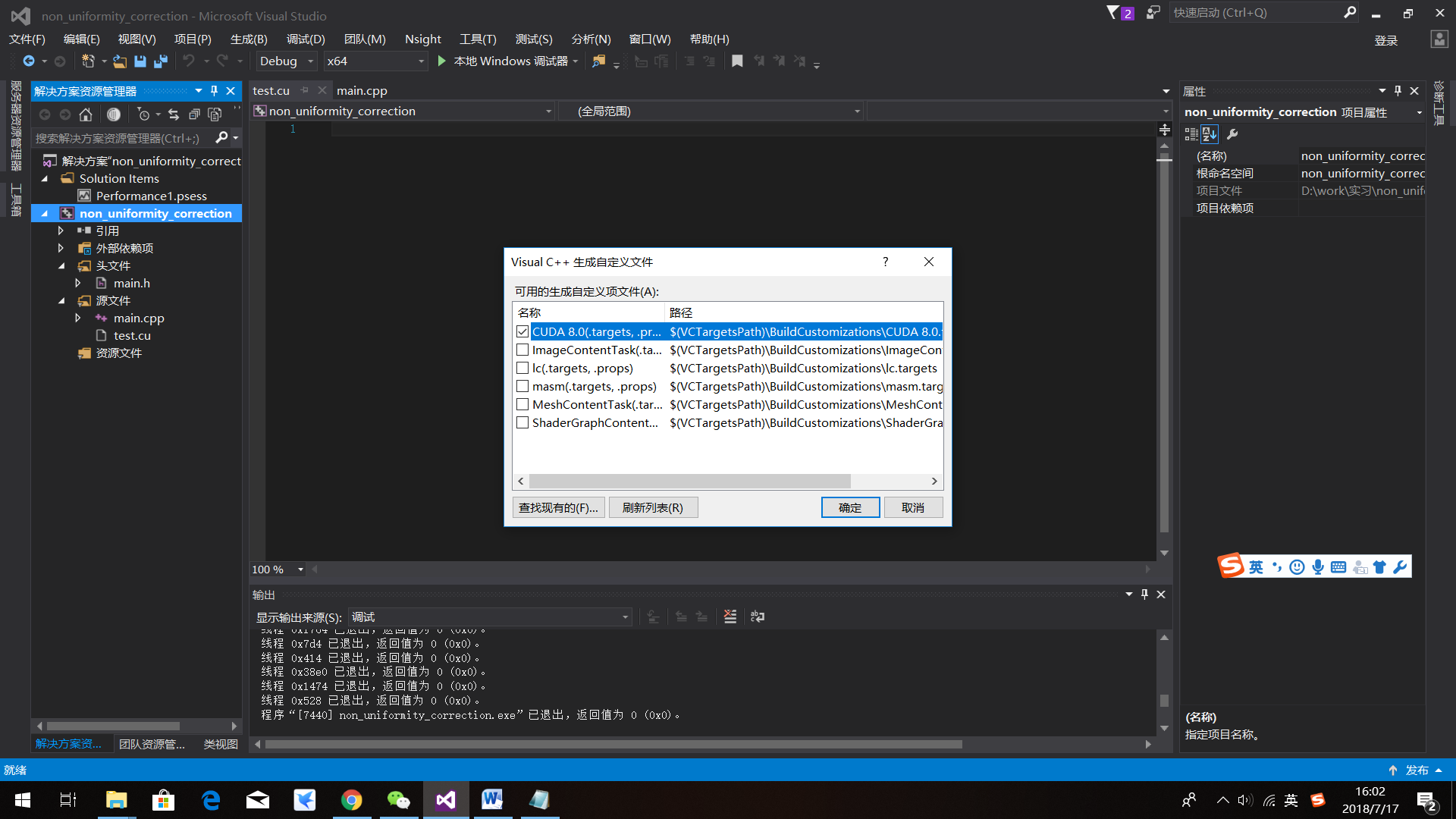
在附加依赖项中包含CUDA和Opencv所有依赖项（静态库）。

Release模式下相同，注意opencv中release模式下的依赖项不带d。具体目录见附带的name\_release.txt和name\_debug.txt。

新建一个test.cu文件

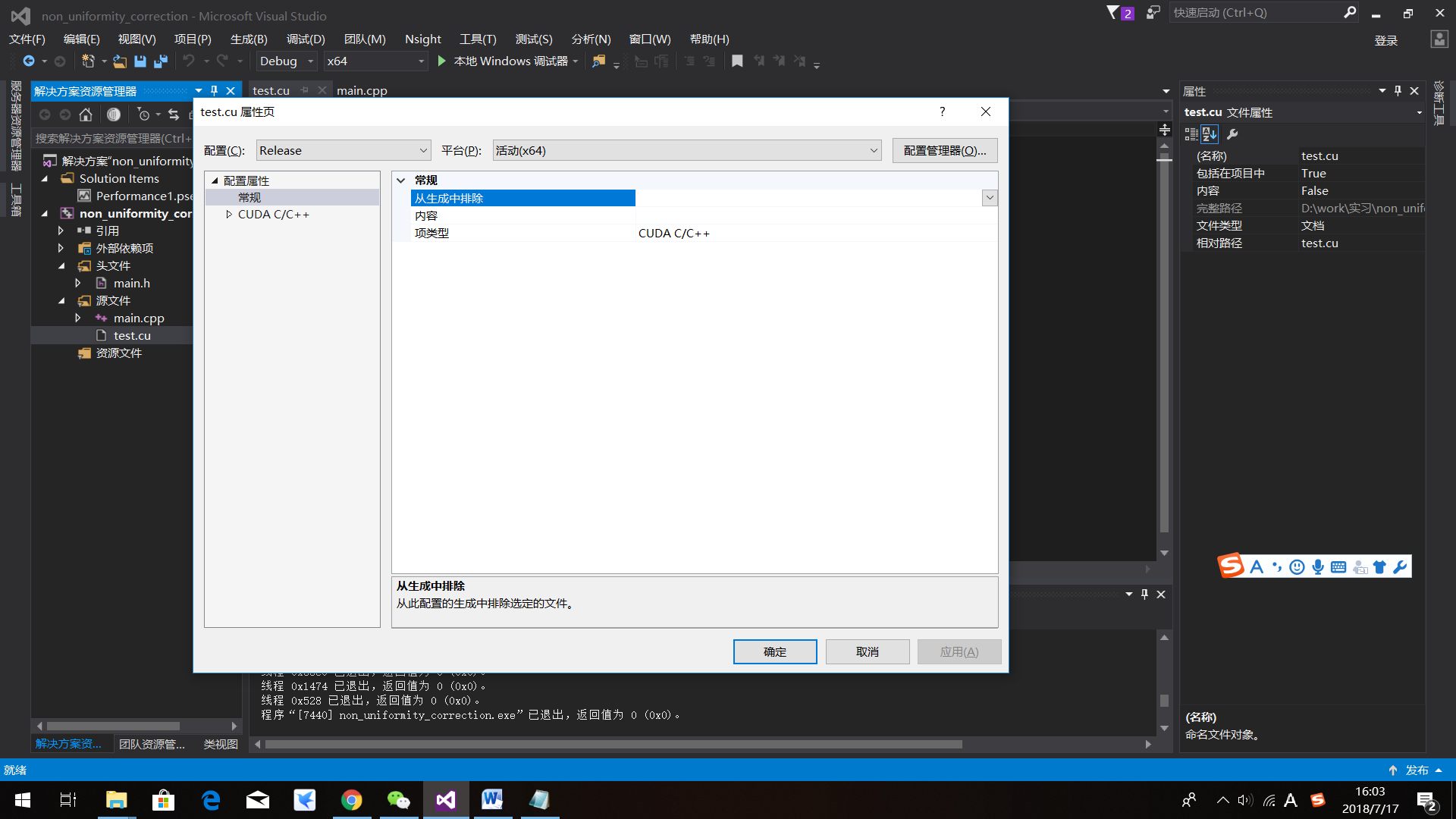


右键项目-》生成依赖项-》生成自定义-》选择cuda生成



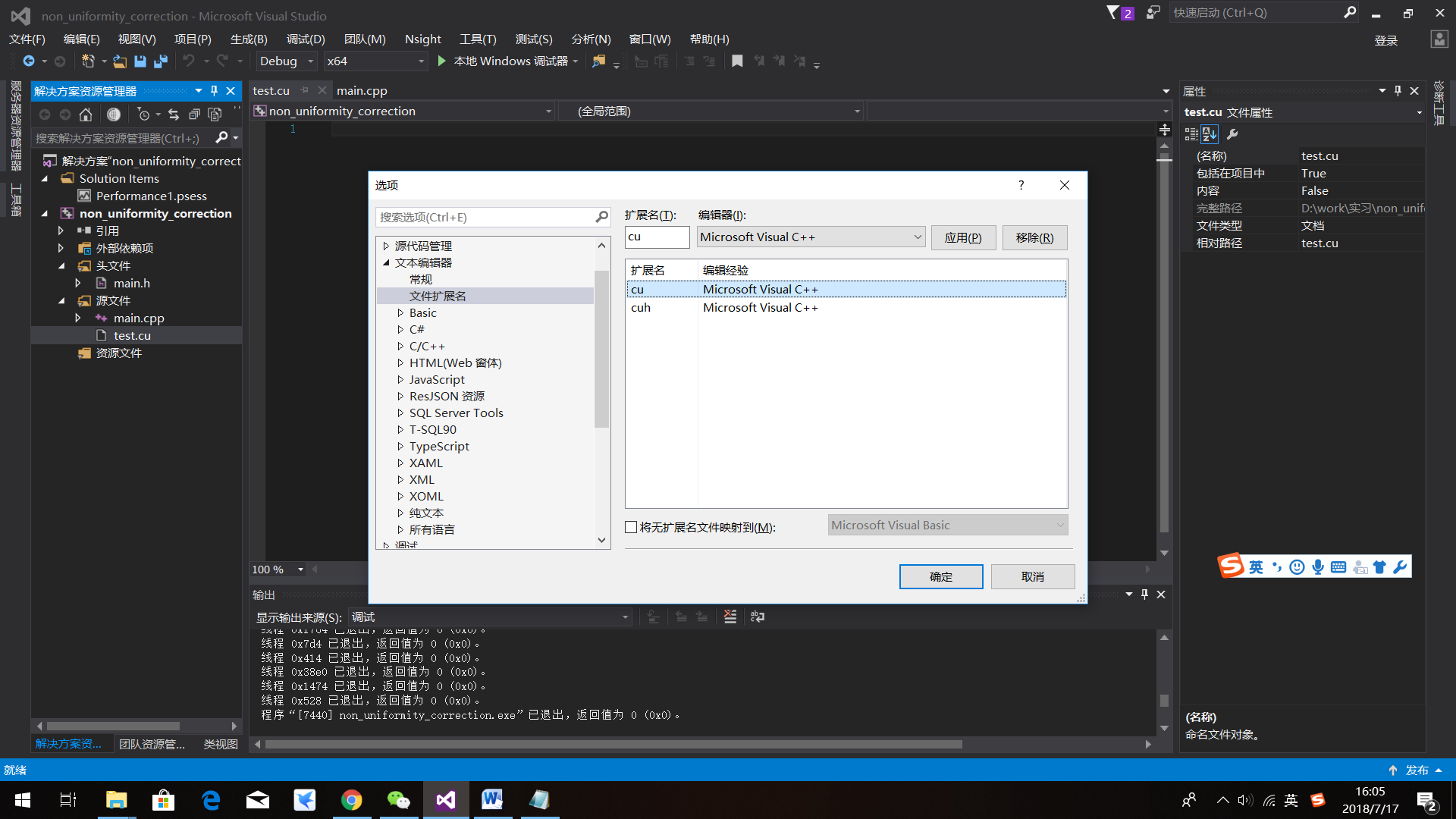
低版本还需要的：

右键.cu文件-》属性-》项类型：CUDA C/C++



在工具-》选项-》文本编辑器-》文件扩展名-》添加cu \cuh两个文件扩展名

这个设置是让VS2010编辑.cu文件时，把.cu文件里的C/C++语法高亮。



**至此配置全部完成**

1. 对于OpenCV+CUDA的简单介绍

<https://www.cnblogs.com/dwdxdy/p/3528711.html>

OpenCV与CUDA结合一般有三种方式

第一种为直接使用OpenCV封装好的带有GPU加速功能的模块，可以满足大部分常规需要。

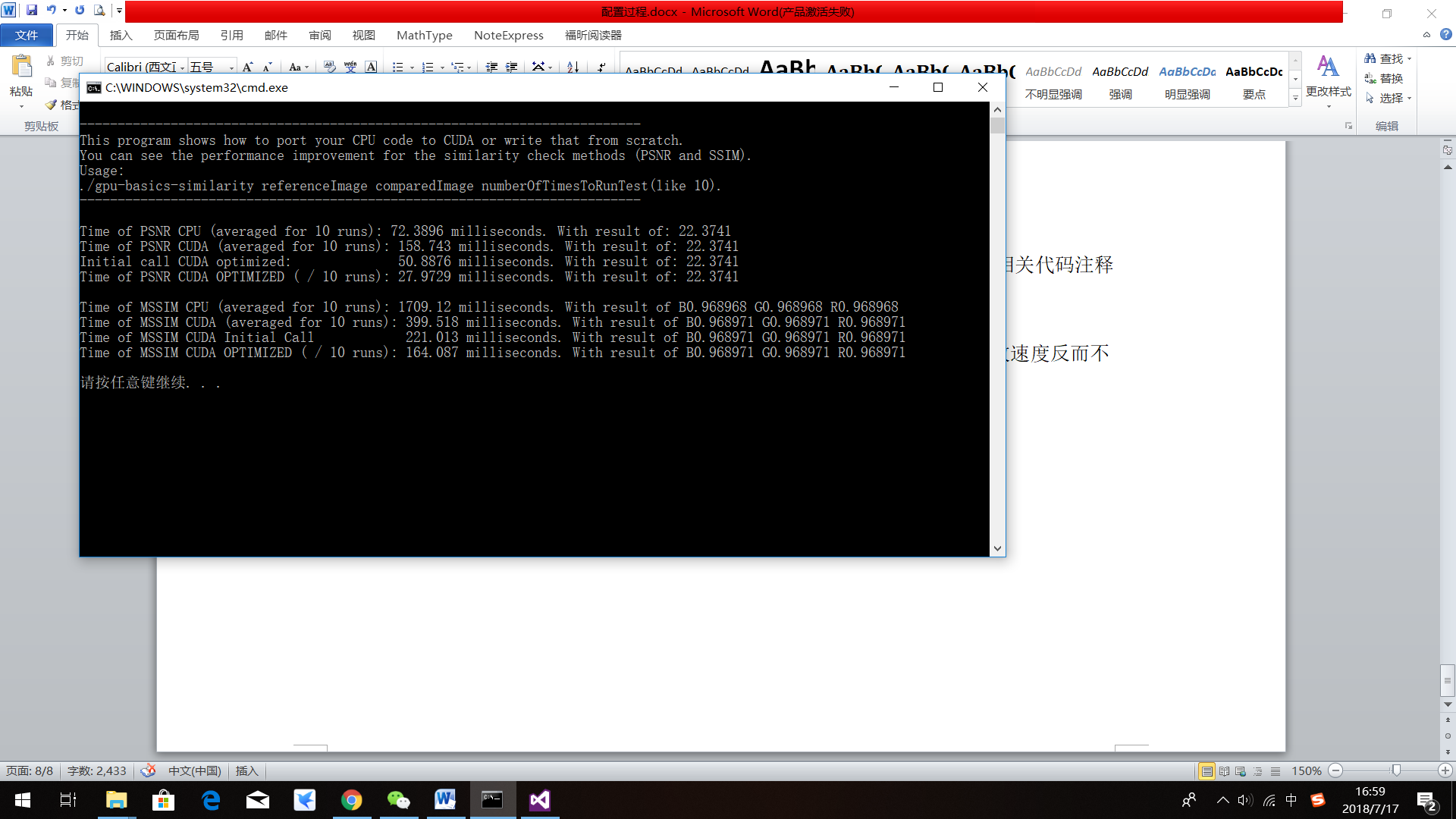
第二种为单独使用CUDA API进行编程，可以全部由用户定义，但是开发成本比较大。

第三章为利用OpenCV中的接口结合CUDA API进行编程，利用OpenCV的接口简化基础操作，同时利用CUDA API编写内核函数，相对比较灵活。

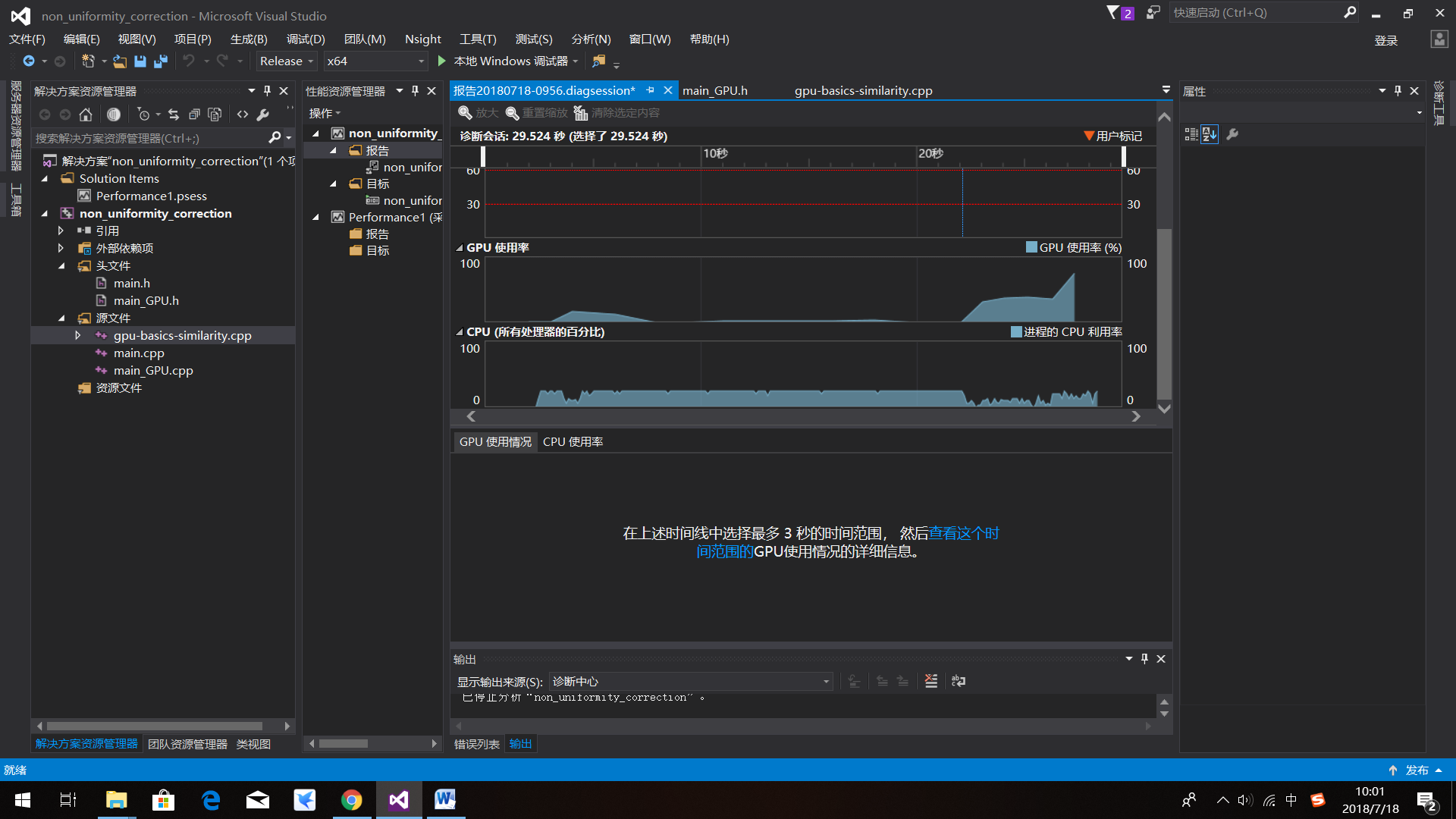
1. 实验测试
   1. 例程（单独使用opencv封装的CUDA库）

测试opencv/source/samples/cpp/gpu/gpu-basics-similarity中的例子。

此例子通过对GPU进行不同的方法进行调用比较说明了进行GPU相关编程时需要注意的一些问题。Main函数中需要输入两幅相同尺寸及深度的图Mat1 及 Mat2 ，同时需要把sstr相关代码注释掉，结果是计算两幅图之间的峰值信噪比以及相似性。



例程中两种指标内第一行为CPU平均计算时间，第二行为没有优化过的GPU平均计算时间，第三行为优化过的GPU程序第一次运行所耗的时间，最后一行为优化过的GPU程序平均计算时间。可以看到GPU加速效果还是比较明显，但是没有优化过的GPU反而会使得计算时间变长。下图为运算过程中CPU和GPU的运算过程。



* + 1. 一些针对例程中的简单的opencv\_GPU编程优化说明

<https://docs.opencv.org/2.4/doc/tutorials/gpu/gpu-basics-similarity/gpu-basics-similarity.html#gpubasicssimilarity>

需要注意的是这是2.4.13.7版本的说明，OpenCV 3.X中不再使用cv::gpu空间，全部改用cv::cuda，3.4.0版本的例程中函数的编写已经与该文档不一致，但是相关的优化思路还是一致的。

3.4.0版本的函数说明：

<https://docs.opencv.org/3.4.0/d1/d1e/group__cuda.html>

* 1. 使用OpenCV API与CUDA API混合编程

opencv/source/samples/cpp/gpu/gpu-thrust-interop的例子

尚未进行过测试

* 1. 另外还有一些基础的sample，可以直接在此基础上进行修改：

Sample地址：opencv/source/samples /gpu

* 1. 总结一下几个使用OpenCV自带API进行GPU编程中需要注意的问题（来自OpenCV官方Doc）
     1. 尽量减少CPU与GPU之间的通信，开销比较小的函数不建议放入GPU中
     2. 切勿将CPU中的Mat与GPU中的GpuMat相互混合引用
     3. 不要将图像的所有通道一起放入函数中，GpuMat只接受单通道或者四通道的图像，三通道的图像建议先进行split拆分或者reshape
     4. GPU中分配空间的开销很大，如果需要多次分配空间请尽可能预先声明结构体，在函数的开头将其实例化以便单次分配完成。这样GPU只会在图像类型或者尺寸改变时重新对图像分配空间。值得注意的是，在具体操作时只需要预先声明即可。
     5. 尽可能避免不必要的数据传输，在运算过程中尽可能不要构造出新的Object，最典型的例子是OpenCV在重载适用于Mat类型的算术运算符 “+”, “-”, “\*”, “^” 以及赋值运算符“=”等符号在运算过程中会构造隐藏的Object存储中间数据。为了避免这种情况，清尽可能使用cuda空间下的函数比如cuda::add, cuda::multiply等函数。
     6. 一般来说cuda空间中的函数默认仍是等待调用完成并返回结果的.如果条件允许的话，尽可能使用异步调用（cuda::Stream）。需要注意的是若对于多Stream操作，需要在一次操作后更新数据再进行下一次操作，使用异步调用是会存在风险的，但是对于不同的操作，他们各自有各自的寄存器空间，因而不会存在这种风险。

<https://stackoverflow.com/questions/17842827/how-to-use-gpustream-in-opencv>

注意异步操作Stream结束之后需要调用stream.waitForCompletion()。