HOW TO BUILD OPENCV FOR WINDOWS WITH CUDA

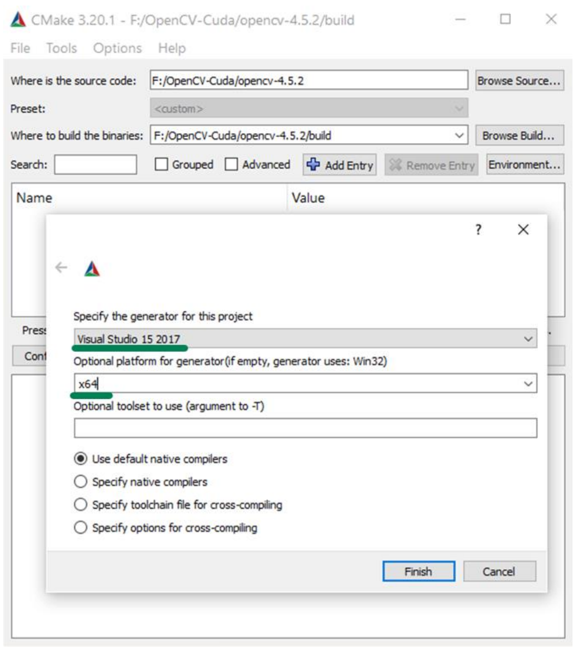
- Windows 10 Operating System
- Visual Studio 2017 Community or Professional
- CUDA Toolkit 11.3
- cuDNN 8.1.1.33 (Please select cudnn’s version according to cuda toolkit version)
- CMake 3.20.1
- opencv-4.5.2 and opencv-contrib-4.5.2

[Generating Visual Studio Project via CMake]

- Creating ‘build’ folder in opencv, and execute cmake-gui

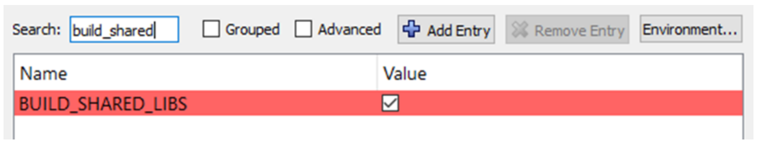
```
mkdir build && cd build && cmake-gui ..
```

- Configuring Options

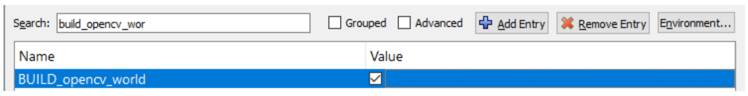

CMake 3.20.1 - F:/OpenCV-Cuda/opencv-4.5.2/build
File Tools Options Help
Browse Source...
Browse Build...
Grouped Advanced Add Entry Remove Entry Environment...
Name Value
Finish Cancel

Where is the source code: F:/OpenCV-Cuda/opencv-4.5.2
Preset: <custom>
Where to build the binaries: F:/OpenCV-Cuda/opencv-4.5.2/build

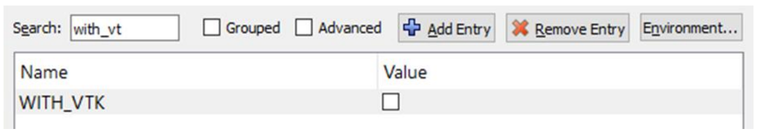
Specify the generator for this project: Visual Studio 15 2017
Optional platform for generator(if empty, generator uses: Win32): x64
Optional toolset to use (argument to -T)
[✓ Use default native compilers]
Specify native compilers
Specify toolchain file for cross-compiling
Specify options for cross-compiling



Search: build_shared
BUILD_SHARED_LIBS ✓

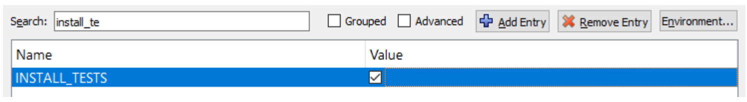


Search: build_opencv_wor
BUILD_opencv_world ✓



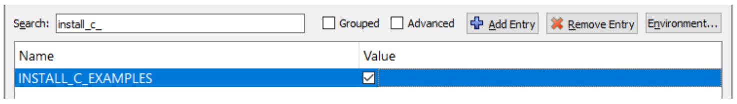
Search: with_vt

Name	Value
WITH_VTK	☐



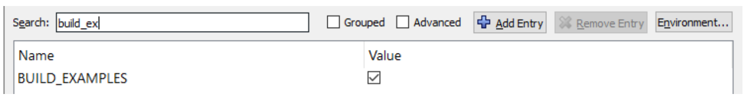
Search: install_te

Name	Value
INSTALL_TESTS	✓



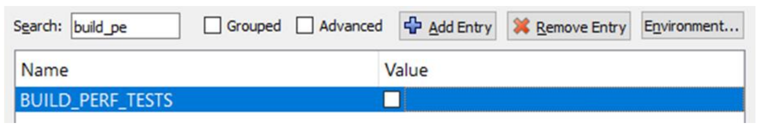
Search: install_c_

Name	Value
INSTALL_C_EXAMPLES	✓



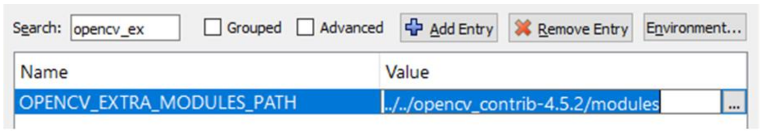
Search: build_ex

Name	Value
BUILD_EXAMPLES	✓



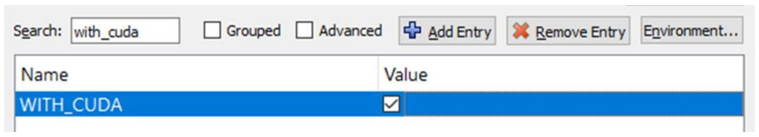
Search: build_pe

Name	Value
BUILD_PERF_TESTS	☐



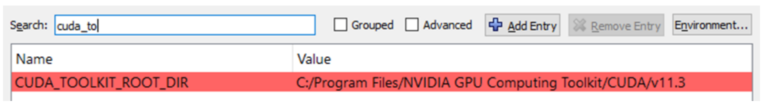
Search: opencv_ex

Name	Value
OPENCV_EXTRA_MODULES_PATH	../../opencv_contrib-4.5.2/modules



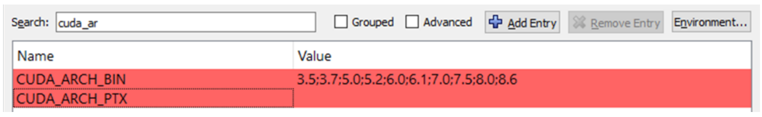
Search: with_cuda

Name	Value
WITH_CUDA	✓



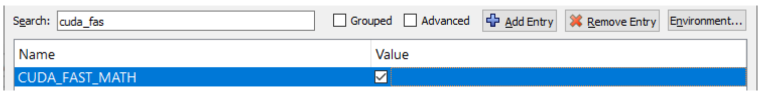
Search: cuda_to

Name	Value
CUDA_TOOLKIT_ROOT_DIR	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.3



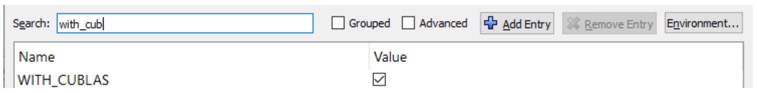
Search: cuda_ar

Name	Value
CUDA_ARCH_BIN	3.5;3.7;5.0;5.2;6.0;6.1;7.0;7.5;8.0;8.6
CUDA_ARCH_PTX	



Search: cuda_fas

Name	Value
CUDA_FAST_MATH	✓

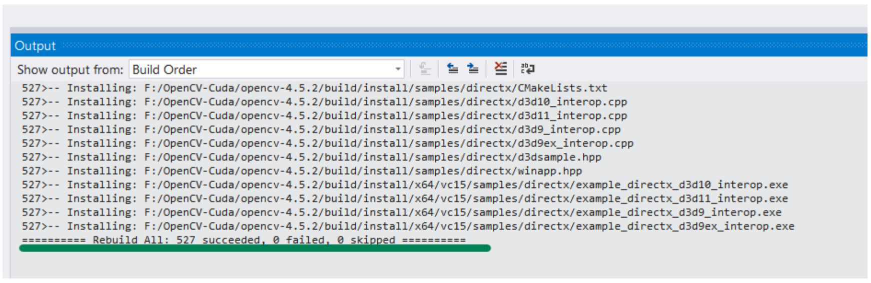


Search: with_cubl

Name	Value
WITH_CUBLAS	✓

- Configuring and Generating on Cmake

[Build on Visual Studio]

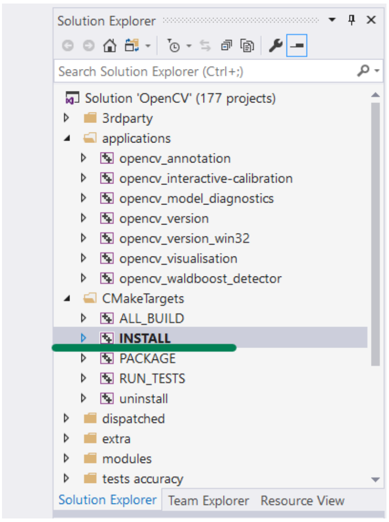


```
527>-- Installing: F:/OpenCV-Cuda/opencv-4.5.2/build/install/samples/directx/CMakeLists.txt
527>-- Installing: F:/OpenCV-Cuda/opencv-4.5.2/build/install/samples/directx/d3d10_interop.cpp
527>-- Installing: F:/OpenCV-Cuda/opencv-4.5.2/build/install/samples/directx/d3d11_interop.cpp
527>-- Installing: F:/OpenCV-Cuda/opencv-4.5.2/build/install/samples/directx/d3d9_interop.cpp
527>-- Installing: F:/OpenCV-Cuda/opencv-4.5.2/build/install/samples/directx/d3d9ex_interop.cpp
527>-- Installing: F:/OpenCV-Cuda/opencv-4.5.2/build/install/samples/directx/d3dsample.hpp
527>-- Installing: F:/OpenCV-Cuda/opencv-4.5.2/build/install/samples/directx/winapp.hpp
527>-- Installing: F:/OpenCV-Cuda/opencv-4.5.2/build/install/x64/vc15/samples/directx/example_directx_d3d10_interop.exe
527>-- Installing: F:/OpenCV-Cuda/opencv-4.5.2/build/install/x64/vc15/samples/directx/example_directx_d3d11_interop.exe
527>-- Installing: F:/OpenCV-Cuda/opencv-4.5.2/build/install/x64/vc15/samples/directx/example_directx_d3d9_interop.exe
527>-- Installing: F:/OpenCV-Cuda/opencv-4.5.2/build/install/x64/vc15/samples/directx/example_directx_d3d9ex_interop.exe
========== Rebuild All: 527 succeeded, 0 failed, 0 skipped ==========
```

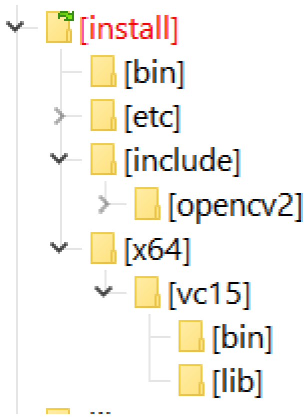
[Installation]

Although the solution was built, you could' t find where header files are.

After built the entire solution, under the "opencv" solution strcutre, there is a folder called "CMakeTargets", expand this folder, you can see "INSTALL" project, and right-click this project, then select "build" option, then after installation, all libraries and head files will be located at the correct path.



When the compilation process is finished, the binaries should be under the **build\install** directory.



In my case, it takes hours to build and install. (PC : Core i7 CPU, 24G RAM)

[Available Configuring Options]

1- BUILD_CUDA_STUBS: This flag is going to build CUDA stubs if there is no CUDA SDK present in the system.

2- BUILD_DOCS: This flag is used to create build rules for the OpenCV documentation.

3- BUILD_EXAMPLES: This flag is used to build all the examples present in the OpenCV library. You can find those examples under the samples folder of the extracted zip on OpenCV.

4- BUILD_ZLIB: This will build the ZLIB from source. The zlib compression utility is a general purpose utility that is useful for various image formats other than PNG, therefore, it can be used without having libpng.

5- BUILD_TIFF: This flag is going to build libtiff from source. It is a library for reading and writing Tagged Image File Format files. It is similar to other libraries namely *libpng* and *libjpeg* which is used to work with *.tiff* format.

6- BUILD_JASPER: This will build the library libjasper from source. It is a project to create a reference implementation of the encoding and decoding specified in the JPEG-2000 Part-1. In other words, it helps in coding and manipulation of images. This library can handle image data in a variety of formats.

7- BUILD_JPEG: This will build the library libjpeg from source. This is a free library for handling JPEG Image Data format. It has got various utilities such as *cjpeg* and *djpeg* to convert the JPEG data into other image file formats. It also has *rdjpgcom* and *wrjpgcom* which helps to insert and extract textual comments in JPEG files.

8- BUILD_PNG: This will build libpng from source. This library is used to handle PNG format images. It is platform independent and is written in C language. It uses zlib for compression and decompression of PNG files.

9- BUILD_WITH_DEBUG_INFO: This flag option in going to include debug info into the release binaries.

10- BUILD_WITH_STATIC_CRT: This flag enables the use of statically linked C-Run Time libraries for statically linked OpenCV. This flag can be enabled in windows system if Microsoft Visual C++ is used for the compilation.

11- BUILD_ANDROID_SERVICE: This flag allows to build OpenCV manager for

Google Play.

12- BUILD_JAVA: This flag will enable the Java Support for OpenCV.

13- BUILD_FAT_JAVA_LIB: This flag will create Java wrapper function i.e. exporting all functions of OpenCV library but it also requires static build of OpenCV modules.

14- INSTALL_C_EXAMPLES, INSTALL_PYTHON_EXAMPLES, INSTALL_ANDROID_EXAMPLES : This flag will enables the installation of the C, Python and Android examples present in the sample folder created during the extraction of the OpenCV zip.

15- INSTALL_TESTS: This will install accuracy and performance test binaries and test data.

16- ENABLE_CCACHE: This flag is known as Compiler Cache flag. This will store the output of the compiler after the compilation of your program. If the file remains unchanged and recompilation is done then the output of compilation will be directly taken from the ccache.

17- ENABLE_PRECOMPILED_HEADERS: This flag is used to enable the use of precompiled headers. Precompiled headers are the header files that are compiled into an intermediate form which are faster to process for the compiler.

18- ENABLE_SOLUTION_FOLDER: This flag enables the use of Solution Folder in Visual Studio IDE. This folder is the container of the projects which you want to do using OpenCV.

19- ENABLE_PROFILING: This flag enables the profiling in the GCC compiler i.e. it add flags **-g, -pg.** Profiling is an aspect of software programming where a programmer can determine the parts in the program code that are time consuming and need to be re-written.

20- OPENCV_GENERATE_PKGCONFIG: This will generate .pc file for pkg-config build tools. pkg-config is a tool used during the compilation of application and libraries. It will help you to provide correct compiler options on the command line i.e. it will find installed libraries which are required for the compilation of software.

21- CV_ENABLE_INTRINSICS: This will enable the compiler to use intrinsic functions which are provided in the programming language to optimize the code.

This optimization will be done when the program requests explicit optimization i.e. when the flag is enabled.

22- **ENABLE_LTO:** This flag will be used to optimize the code during the linking of libraries. The linker will pull all the object files and combine them to optimize the code.

23- **ENABLE_THIN_LTO:** This flag will be used for optimization of program in incremental and scalable manner. It is similar to regular LTO as clang will emit LLVM bitcode after the compilation. The ThinLTO bitcode is augmented with a compact summary of the module. During the link step, only the summaries are read and merged into a combined summary index, which includes an index of function locations for later cross-module function importing. Fast and efficient whole-program analysis is then performed on the combined summary index.

24- **LAPACK_LIBRARIES:** LAPACK (Linear Algebra PACKage) is written in Fortran 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The original goal of the LAPACK project was to make the widely used EISPACK and LINPACK libraries run efficiently on shared-memory vector and parallel processors.

25- **MKL_ROOT_DIR:** Intel® Math Kernel Library (Intel® MKL) optimizes code with minimal effort for future generations of Intel® processors. It is compatible with your choice of compilers, languages, operating systems, and linking and threading models. It uses industry-standard C and Fortran APIs for compatibility with popular BLAS, LAPACK, and FFTW functions for its working.

26- **OPENCV_EXTRA_MODULES_PATH:** This flag is used to specify the path of the extra modules which will be installed along with the installation of OpenCV. Note that, we have downloaded ***opencv_contrib*** in step 2 which is nothing but the extra modules in OpenCV. You can set the path on this flag to the ***opencv_contrib*** downloaded folder.

27- **OPENCV_GENERATE_PKGCONFIG:** This flag is used to generate the opencv4.pc file. The pkg-config provides command line options for compiling the programs i.e. pkg-config finds the path of installed libraries automatically which are going to be used in your program. OpenCV 4.1.0 doesn’t generate the opencv4.pc file unless this flag is specified.

28- **WITH_CUDA:** This option allows the CUDA support in the OpenCV. You need to make sure that you have CUDA enabled GPU in your machine in order to use the CUDA library. CUDA is generally used to make the computations faster with the help of GPU.

29- **WITH_DIRECTX:** This option allows the Microsoft DirectX support for OpenCV. It is a collection of application programming interfaces (APIs) for handling tasks related to multimedia, especially game programming and video on Microsoft platforms.

30- **WITH_EIGEN:** This flag enables the Eigen library support for OpenCV. It is a high-level C++ library of template headers for linear algebra matrix and vector operations, geometrical transformations, numerical solvers and related algorithms.

31- **WITH_GDAL:** The Geospatial Data Abstraction Library (GDAL) is a computer software library for reading and writing raster and vector geospatial data formats, and is released under the permissive X/MIT style free software license by the Open Source Geospatial Foundation. As a library, it presents a single abstract data model to the calling application for all supported formats. It may also be built with a variety of useful command line interface utilities for data translation and processing.

32- **WITH_GSTREAMER:** GStreamer is a library for constructing graphs of media-handling components. The applications it supports range from simple Ogg/Vorbis playback, audio/video streaming to complex audio (mixing) and video (non-linear editing) processing.

33- **WITH_HALIDE:** Halide is an open-source project that let us write image processing algorithms in well-readable format, schedule computations according to specific device and evaluate it with a quite good efficiency.

34- **WITH_HPX:** HPX (High Performance ParalleX) is a general purpose C++ runtime system for parallel and distributed applications of any scale. This library strictly adheres to the C++11 Standard and leverages the Boost C++ Libraries which makes HPX easy to use, highly optimized, and very portable. HPX is developed for conventional architectures including Linux-based systems, Windows, Mac, and the BlueGene/Q, as well as accelerators such as the Xeon Phi.

35- **WITH_OPENGL:** This flag will allow the OpenGL support is OpenCV for building application related to OpenGL. OpenGL is a library which is used to render 3D scenes. These 3D scenes can be later processed using OpenCV library in the form of images and videos.

36- **WITH_TBB:** The TBB flag makes use of Intel's Parallel Programming and Heterogeneous Computing library i.e. Thread Building Block library. The library provides a wide range of features for parallel programming, including generic parallel algorithms, concurrent containers, a scalable memory allocator, work-stealing task scheduler, and low-level synchronization primitives. If you want to know more about this library click on this link.

37- **WITH_OPENCL:** OpenCL (Open Computing Language) is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators

38- **WITH_QT:** Qt is a cross-platform application development framework for desktop, embedded and mobile. Qt is not a programming language on its own. It is a framework written in C++. With Qt, GUIs can be written directly in C++ using its Widgets modules. Qt also comes with an interactive graphical tool called ***Qt Designer*** with functions as a code generator for Widgets based GUIs.