Table of Contents

[**OpenCV with CUDA and Qt Creator on Windows (MSVC): Full Manual** 2](#_Toc180358526)

[**1. Installing Necessary Tools** 2](#_Toc180358527)

[**1.1 Install Visual Studio** 2](#_Toc180358528)

[**1.2 Install Qt and Qt Creator** 2](#_Toc180358529)

[**1.3 Install CUDA Toolkit** 2](#_Toc180358530)

[**1.4 Install cuDNN (for deep learning acceleration)** 2](#_Toc180358531)

[**1.5 Install CMake** 2](#_Toc180358532)

[**2. Downloading OpenCV and opencv\_contrib** 2](#_Toc180358533)

[**2.1 Download OpenCV Source** 2](#_Toc180358534)

[**2.2 Download opencv\_contrib** 2](#_Toc180358535)

[**3. Building OpenCV with CMake and CUDA Support** 3](#_Toc180358536)

[**3.1 Open CMake GUI** 3](#_Toc180358537)

[**3.2 Configure CMake for OpenCV with CUDA and Qt** 3](#_Toc180358538)

[**3.3 Build OpenCV with CUDA and Qt in Visual Studio** 3](#_Toc180358539)

[**4. Configuring Qt Project to Use OpenCV** 3](#_Toc180358540)

[**4.1 Add OpenCV to the System Path** 3](#_Toc180358541)

[**4.2 Setting Up the .pro File in Qt Creator** 4](#_Toc180358542)

[**5. Testing GPU Usage in Qt Project** 4](#_Toc180358543)

[**5.1 Example Code to Verify GPU Acceleration** 4](#_Toc180358544)

[**5.2 Running the Project with GPU** 5](#_Toc180358545)

[**6. Command to Check GPU Status** 5](#_Toc180358546)

[**Conclusion** 6](#_Toc180358547)

**OpenCV with CUDA and Qt Creator on Windows (MSVC): Full Manual**

**1. Installing Necessary Tools**

**1.1 Install Visual Studio**

* Download **Visual Studio 2022** from the [Visual Studio website](https://visualstudio.microsoft.com/).
* During installation, select the **Desktop development with C++** workload, ensuring that **MSVC** compiler and **CMake** support are installed.

**1.2 Install Qt and Qt Creator**

* Download the **Qt installer** from Qt's official website.
* Install **Qt 6.8.0** (or a version that matches your MSVC installation) and choose the **MSVC 2022 (64-bit)** toolchain during installation.

**1.3 Install CUDA Toolkit**

* Download and install the **CUDA Toolkit** from [NVIDIA's website](https://developer.nvidia.com/cuda-toolkit). Choose a version that matches your system and GPU.
* Ensure that the necessary **CUDA drivers** and **tools** are installed.

**1.4 Install cuDNN (for deep learning acceleration)**

* Download the **cuDNN library** from [NVIDIA's cuDNN page](https://developer.nvidia.com/cudnn) and extract the contents.
* Copy the bin, include, and lib files from the cuDNN package into your **CUDA installation directory** (e.g., C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.x).

**1.5 Install CMake**

* Download and install **CMake** from cmake.org.
* CMake is used to configure and build OpenCV from source with CUDA and Qt support.

**2. Downloading OpenCV and opencv\_contrib**

**2.1 Download OpenCV Source**

* Download **OpenCV** from the OpenCV releases page.
* Extract it to a folder, for example, C:\opencv\sources.

**2.2 Download opencv\_contrib**

* Download the **opencv\_contrib** modules from the [opencv\_contrib GitHub repository](https://github.com/opencv/opencv_contrib).
* Extract or clone it to a directory like C:\Users\<username>\opencv\_contrib.

**3. Building OpenCV with CMake and CUDA Support**

**3.1 Open CMake GUI**

1. **Where is the source code**: Set this to the OpenCV source folder (e.g., C:/opencv/sources).
2. **Where to build the binaries**: Set this to an empty build folder (e.g., C:/opencv-build).

**3.2 Configure CMake for OpenCV with CUDA and Qt**

1. Click **Configure** and choose **Visual Studio 2022** as the generator and **x64** as the platform.
2. Enable the following options in CMake:
   * **WITH\_CUDA**: **ON** (for CUDA support).
   * **WITH\_CUBLAS**: **ON** (CUDA’s basic linear algebra support).
   * **WITH\_CUDNN**: **ON** (for cuDNN support).
   * **WITH\_QT**: **ON** (for Qt integration).
   * **WITH\_OPENGL**: **ON** (for OpenGL support).
   * **OPENCV\_EXTRA\_MODULES\_PATH**: Set this to C:/Users\<username>\opencv\_contrib/modules (path to opencv\_contrib modules).
   * **CMAKE\_INSTALL\_PREFIX**: Set this to C:/opencv-build/install (where OpenCV will be installed).
3. Click **Configure** again to refresh the settings, and **Generate** the project files.

**3.3 Build OpenCV with CUDA and Qt in Visual Studio**

1. After generating the files, click **Open Project** to open the project in **Visual Studio**.
2. In Visual Studio, select the **Release** mode.
3. Right-click the solution and select **Build Solution**. This will compile OpenCV with CUDA and Qt support.
4. After the build is complete, right-click the solution and select **INSTALL** to copy the build files to the CMAKE\_INSTALL\_PREFIX directory (e.g., C:/opencv-build/install).

**4. Configuring Qt Project to Use OpenCV**

**4.1 Add OpenCV to the System Path**

To ensure your program can find OpenCV’s .dll files at runtime, add the OpenCV bin directory to your **Windows PATH**:

1. Open **System Properties** > **Advanced System Settings** > **Environment Variables**.
2. Under **System variables**, find Path, select it, and click **Edit**.
3. Add C:/opencv-build/install/x64/vc17/bin to the Path.

**4.2 Setting Up the .pro File in Qt Creator**

In your Qt project's **.pro file**, add the following lines to link OpenCV with CUDA and Qt:

pro

Copy code

QT += core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++17

# OpenCV include directory

INCLUDEPATH += C:/opencv-build/install/include

# OpenCV library directory and linking the necessary libraries

LIBS += -LC:/opencv-build/install/x64/vc17/lib \

-lopencv\_core4100 \

-lopencv\_imgproc4100 \

-lopencv\_highgui4100 \

-lopencv\_imgcodecs4100 \

-lopencv\_cudaarithm4100 \

-lopencv\_cudafilters4100 \

-lopencv\_cudaimgproc4100 \

-lopencv\_dnn4100

# Replace '4100' with the correct OpenCV version you built (e.g., 4100 for OpenCV 4.10.0)

**5. Testing GPU Usage in Qt Project**

**5.1 Example Code to Verify GPU Acceleration**

Here’s a simple Qt project code that tests whether GPU is being used and compares performance between the CPU and GPU:

cpp

Copy code

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

#include <opencv2/highgui.hpp>

#include <opencv2/core/cuda.hpp>

#include <iostream>

int main() {

// Check for CUDA-capable devices

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

if (num\_devices == 0) {

std::cerr << "No CUDA-capable devices detected." << std::endl;

return -1;

}

cv::Mat image = cv::imread("C:/Users/skyli/OneDrive - Universiti Teknologi Malaysia (UTM)/Documents/OpenCVMSVC/image.jpg", cv::IMREAD\_COLOR);

if (image.empty()) {

std::cerr << "Image not found!" << std::endl;

return -1;

}

// Measure GPU time

cv::cuda::GpuMat d\_image, d\_gray;

int64 gpu\_start = cv::getTickCount();

d\_image.upload(image); // Upload image to GPU

cv::cuda::cvtColor(d\_image, d\_gray, cv::COLOR\_BGR2GRAY); // Convert to grayscale on GPU

cv::Mat gray\_gpu;

d\_gray.download(gray\_gpu); // Download back to CPU

int64 gpu\_end = cv::getTickCount();

double gpu\_time = (gpu\_end - gpu\_start) / cv::getTickFrequency();

std::cout << "GPU processing time: " << gpu\_time << " seconds" << std::endl;

// Measure CPU time

int64 cpu\_start = cv::getTickCount();

cv::Mat gray\_cpu;

cv::cvtColor(image, gray\_cpu, cv::COLOR\_BGR2GRAY); // Convert to grayscale on CPU

int64 cpu\_end = cv::getTickCount();

double cpu\_time = (cpu\_end - cpu\_start) / cv::getTickFrequency();

std::cout << "CPU processing time: " << cpu\_time << " seconds" << std::endl;

// Display the results

cv::imshow("Gray Image (GPU)", gray\_gpu);

cv::imshow("Gray Image (CPU)", gray\_cpu);

cv::waitKey(0);

return 0;

}

**5.2 Running the Project with GPU**

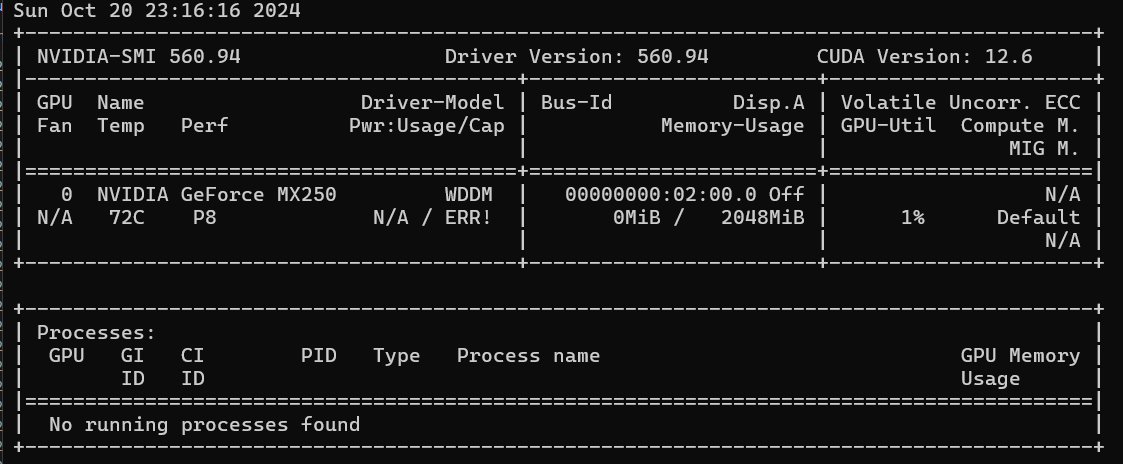
1. **Run the Qt project** in **Release mode**.
2. Check the **console output** for:
   * Number of **CUDA-capable devices** detected.
   * **GPU processing time** vs. **CPU processing time**.

**6. Command to Check GPU Status**

Use **nvidia-smi** in the Command Prompt to check GPU status: nvidia-smi

This will display the current status of your NVIDIA GPU, the driver version, and any processes currently using the GPU.

Example:



The presence of your **Qt application** in the Processes section confirms that the GPU is being used.

**Conclusion**

* This manual guides you through **installing and configuring OpenCV** with **CUDA** and **Qt** on Windows using **MSVC**.
* The example provided ensures that **GPU acceleration** is used and verifies performance improvements over the CPU.
* You can check **GPU utilization** using nvidia-smi or by observing performance timing differences between GPU and CPU.